# Towards Evaluating and Understanding the Adversarial Robustness of Random Transformation Defenses

*Zachary Golan-Strieb*
*David A. Wagner*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-241
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-241.html

December 1, 2021

# Towards Evaluating and Understanding the Adversarial Robustness of Random Transformation Defenses

by

Zachary Golan-Strieb

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor David Wagner
Assistant Professor Jacob Steinhart

Spring 2021

The dissertation of Zachary Golan-Strieb, titled Towards Evaluating and Understanding the Adversarial Robustness of Random Transformation Defenses, is approved:

[illegible] Date 5/11/2021

[illegible] Date 05/04/2021

University of California, Berkeley

# Towards Evaluating and Understanding the Adversarial Robustness of Random Transformation Defenses

Zachary Golan-Strieb
UC Berkeley
Berkeley, CA, USA
zacharyjgs@berkeley.edu

David A. Wagner
UC Berkeley
Berkeley, CA, USA
daw@cs.berkeley.edu

## Abstract

*Current machine learning models suffer from evasion attacks such as adversarial examples. This introduces security and safety concerns that lack any clear solution. Recently, the use of random transformations (RT) has emerged as a promising defense against adversarial examples. However, it has not been rigorously evaluated, and its effectiveness is not well-understood. In this paper, we attempt to construct the strongest possible RT defense through the informed selection of transformations and the use of Bayesian optimization to tune their parameters. Furthermore, we attempt to identify the strongest possible attack to evaluate our RT defense. Our new attack vastly outperforms the naive attack, reducing the accuracy of our model by an additional 30%. In the process of formulating our defense and attack, we perform several ablation studies for both problems, drawing insights that we hope will broadly benefit scientific communities that study stochastic neural networks and robustness properties.*

## 1. Introduction

Today, deep neural networks are widely deployed in safety-critical settings such as autonomous driving and cybersecurity. Despite their effectiveness at solving a wide-range of challenging problems, they are known to have a major vulnerability. Tiny crafted perturbations added to inputs (so called *adversarial examples*) can arbitrarily manipulate the outputs of these large models, posing a threat to the safety and privacy of the millions of people who rely on existing ML systems. The importance of this problem has drawn substantial attention, and yet we have not devised a concrete countermeasure as a research community.

Adversarial training [33] has been the foremost approach for defending against adversarial examples. While adversarial training provides increased robustness, it results in a loss of accuracy on benign inputs. Recently, a promising line of defenses against adversarial examples has emerged. These defenses randomize either the model parameters or the inputs themselves [28, 23, 36, 31, 48, 51, 3, 30, 8, 12, 18]. Introducing randomness into the model can be thought of as a form of smoothing that removes sinuous portions of the decision boundary where adversarial examples frequently lie [22]. Among these randomization approaches, Raff et al. [36] propose Barrage of Random Transforms (BaRT), a new defense which applies a large set of random image transformations to classifier inputs. They report a $24\times$ increase in robust accuracy over previously proposed defenses.

Despite these promising results, the research community still lacks a clear understanding of how to properly evaluate random defenses. This is concerning as a defense can falsely appear more robust than it actually is when evaluated using sub-optimal attacks [1, 43], most of which are developed for deterministic models without input transformations. Therefore, in this work, we improve existing attacks on randomized defenses, and use them to rigorously evaluate BaRT and similar schemes. We find that sub-optimal attacks have led to an overly optimistic view of these random defenses. For instance, we show that BaRT is much less secure than previously thought, formulating a new attack that reduces its security (from 56% to 27% robust accuracy, on Imagenette). Moreover, we demonstrate how to improve random transform defenses and present a new trade-off between clean and robust accuracy (our scheme: 89% clean accuracy, 29% robust accuracy; vs adversarial training: 78% clean accuracy, 37% robust accuracy). To summarize, we make the following contributions:

- We show that non-differentiable transforms impede optimization during an attack and that even the state-of-the-art technique (BPDA [1]) for circumventing non-differentiability is not sufficiently effective.
- We evaluate methods for attacking randomized defenses and identify a new method that results in a large improvement over the current state-of-the-art attack on schemes with random image transformations.
- We explain the success of the attacks through the vari-

ance of their gradients.

- We show how to use Bayesian optimization for hyperparameter tuning to improve this class of defenses.

## 2. Background and Related Works

### 2.1. Adversarial Examples

Adversarial examples are carefully perturbed inputs designed to fool a machine learning model [42, 5, 15]. An adversarial perturbation $\delta$ is typically constrained to be within some $\ell_p$-norm ball with a radius of $\epsilon$. The $\ell_p$-norm ball is a proxy to the "imperceptibility" of $\delta$ and can be thought of as the adversary's budget. In this work, we primarily use $p = \infty$. Finding the worst-case perturbation $\delta^*$ requires solving the following optimization problem:

$$x_{\text{adv}} = x + \delta^* = x + \underset{\delta : \|\delta\|_p \leq \epsilon}{\arg\max}\; L(x + \delta, y) \tag{1}$$

where $L : \mathbb{R}^d \times \mathbb{R}^C \to \mathbb{R}$ is the loss function of the target model which, in our case, is a classifier which makes predictions among $C$ classes. Projected gradient descent (PGD) is often used to solve the optimization problem in Eqn. 1.

### 2.2. Randomization Defenses

A number of recent papers have proposed defenses against adversarial examples which utilize inference-time randomization. One common approach is to sample weights of the network from some probability distribution [30, 23, 31, 3]. In this paper, we instead focus on defenses that apply random transforms to the input [36, 48, 51, 8]. Many of these defenses claim to achieve state-of-the-art robustness. Unlike prior evaluations, we test these defenses using a wide range of white-box attacks as well as a novel stronger attack. A key issue when evaluating these schemes is that PGD attacks require gradients through the entire model pipeline, but many defenses use non-differentiable transforms. As we show later, this can cause evaluation results to be misleading.

Different works have tried applying different random transformations to their inputs. Xie et al. randomly resize and pad images [48]. While this defense ranked second in the NeurIPS 2017 adversarial robustness competition, their security evaluation did not consider adaptive attacks where the adversary has full knowledge of the transformations.

Zhang et al. [51] add Gaussian noise to the input and then quantize it. They report that this defense outperforms all of the NeurIPS 2017 submissions. For their attack, Zhang et al. approximate the gradient of the transform, which could lead to a sub-optimal attack. In this paper, we use the exact gradients for all transformations when available.

More recently, Raff et al. [36] claim to achieve a state-of-the-art robust accuracy 24× better than adversarial training using a random transformation defense known as Barrage of Random Transforms (BaRT). BaRT involves randomly sampling a large set of image transformations and applying them to the input in a random order. Because many transformations are non-differentiable, BaRT evaluates their scheme using an attack that approximates the gradients of the transforms. In Section 4.2, we show that this approximation is ineffective, giving overly optimistic impression of BaRT's robustness, and we re-evaluate BaRT using a stronger attack which utilizes exact transform gradients.

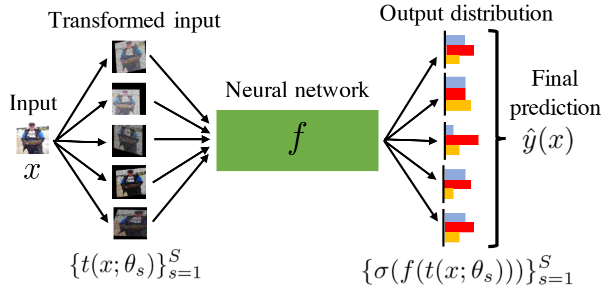


Figure 1: Diagram illustrating components of RT.

## 3. Random Transformation Defense

We focus in this paper on an architecture we call random transformation defense (RT).

### 3.1. Description of RT

RT repeatedly applies a randomly chosen transform to the input, uses a neural network to make a prediction, and then averages the softmax prediction scores:

$$g(x) := \mathbb{E}_{\theta \sim p(\theta)}\left[\sigma\left(f\left(t(x;\theta)\right)\right)\right] \tag{2}$$

where $\sigma(\cdot)$ is the softmax function, $f : \mathbb{R}^d \to \mathbb{R}^C$ a neural network ($C$ is the number of classes), and the transformation $t(\cdot;\theta) : \mathbb{R}^d \to \mathbb{R}^d$ is parameterized by a random variable $\theta$ drawn from some distribution $p(\theta)$.

In practice, we approximate the expectation in Eqn. 2 with $n$ Monte Carlo samples per one input $x$:

$$g(x) \approx g_n(x) := \frac{1}{n}\sum_{i=1}^{n} \sigma\left(f(t(x;\theta_i))\right) \tag{3}$$

We then define the final prediction as the class with the largest softmax probability: $\hat{y}(x) = \arg\max_{c \in [C]}\; [g_n(x)]_c$. Note that this decision rule is different from most previous works that use a majority vote on hard labels, i.e., $\hat{y}_{\text{maj}}(x) = \arg\max_{c \in [C]}\; \sum_{i=1}^{n} \mathbb{1}\left\{c = \arg\max_{j \in [C]}\; f_j(x)\right\}$ [36, 8]. We later show in Appendix E.1 that our rule is empirically superior to the majority vote. From the Law of Large Numbers, as $n$ increases, the approximation in Eqn. 3 converges to the expectation in Eqn. 2. Fig. 1 illustrates the structure and the components of the RT architecture.

In practice $t(\cdot;\theta)$ may represent a sequence of $S$ image transformations, where $\theta = \{\theta^{(1)}, \dots, \theta^{(S)}\}$ and $\theta^{(s)}$ denotes the parameters for the $s$th transformation.

$$t(x;\theta) = t_{\theta^{(S)}} \circ t_{\theta^{(S-1)}} \circ \cdots \circ t_{\theta^{(1)}}(x) \tag{4}$$

Each $\theta^{(s)}$ is a random variable comprised of three components which dictate the properties of a transformation:

1. The *type* $\pi$ of transformation to apply (e.g., rotation, JPEG compression, etc.), which is randomly drawn, without replacement, from a pool of $K$ transformation types.
2. A *boolean* indicating whether the transformation will be applied. This is a Bernoulli random variable with probability $p \in [0,1]$ $(\mathrm{Bern}\,(p))$.
3. The *strength* of the transformation (e.g., rotation angle, JPEG quality, etc.) following a predefined distribution which depends on the transformation type and is parameterized by $\alpha \in [0,1]$.

Specifically, for each of the $n$ RT samples, we sample a permutation of size $S$ out of $K$ transformation types in total, i.e. $\{\pi(1), \dots, \pi(S)\} \in \mathrm{Perm}(K,S)$. Then the boolean and the strength of the $s$-th transform are drawn from the distributions $p_{\pi(s)}$ and $\alpha_{\pi(s)}$. A different set of $S$ transforms are applied to each of the $n$ copies of the input in parallel.

### 3.2. Choices of Transformations

In this work, we use a pool of $K = 33$ different image transformations. Our 33 transformations include 19 differentiable and 2 non-differentiable transforms taken from the 30 BaRT transforms [36] (counting each type of noise injection as its own transform). We replace non-differentiable transformations with a smooth differentiable alternative [39]. The transformations fall into seven groups: noise injection (7 transforms), blur filtering (4 transforms), color-space alteration (8), edge detection (2), lossy compression (3), geometric transformation (5), and stylization (4). All transforms are described in Appendix A.

## 4. Attacks on RT

### 4.1. Adversarial Objective

The problem of finding adversarial examples for an RT model can be framed as the following optimization problem where $f$, $t$, and $p(\theta)$ are as defined in Section 3.1, $(x,y)$ is an image-label pair, and $L : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ outputs a loss given the predicted logits and true label.

$$\max_{\|\delta\|_p \le \epsilon} L\left(\mathop{\mathbb{E}}_{\theta \sim p(\theta)}\left[f\left(t(x+\delta;\theta)\right)\right], y\right) \tag{5}$$

We estimate the expectation by Monte Carlo sampling. We denote this approximate loss as $L_n^{\text{logit}}$:

$$L_n^{\text{logit}}(x,y) \coloneqq L\left(\frac{1}{n}\sum_{i=1}^{n} f\left(t(x+\delta;\theta_i)\right), y\right) \tag{6}$$

Another way to approximate this objective, often referred to as Expectation over Transformation (EoT), is to move the expectation and the average outside of the loss [2].

$$L_n^{\text{eot}}(x,y) \coloneqq \frac{1}{n}\sum_{i=1}^{n} L\left(f\left(t(x+\delta;\theta_i)\right), y\right) \tag{7}$$

$$\approx \mathop{\mathbb{E}}_{\theta \sim p(\theta)}\left[L\left(f\left(t(x+\delta;\theta)\right), y\right)\right] \tag{8}$$

Note that $L_n^{\text{eot}}$ is different from the true objective we are optimizing in Eqn. 5. On the other hand, $L_n^{\text{logit}}$ is a biased estimator of the true objective [37]. We empirically compare these formulations in Section 5.2.

### 4.2. Non-Differentiability and BPDA

To solve Eqn. 5, we need to compute gradients of the approximate objective (either Eqn. 6 or Eqn. 7) with respect to the perturbation $\delta$. This can be computed through backpropagation and requires the Jacobian of the transform $t(\cdot;\theta_i)$ with respect to its input. However, some transformations are innately non-differentiable (e.g., JPEG compression, precision reduction, etc.), rendering first-order methods nonviable.

To circumvent this problem, Raff et al. uses backward-pass differentiable approximation (BPDA) [1] to estimate the "derivatives" of a non-differentiable function by first approximating the function with a neural network and using the derivative of the neural network instead. To approximate a transformation, we train a model $\tilde{t}_\phi$ that minimizes the Euclidean distance between the transformed image and the model output:

$$\min_{\phi} \left\|\tilde{t}_\phi(x;\theta) - t(x;\theta)\right\|_2 \tag{9}$$

We evaluate the BPDA approximation below in a series of experiments that compare the effectiveness of the BPDA attack to an attack that uses exact gradients.

### 4.3. Experimental Setup

Our experiments use Imagenette [24], a ten-class subset of ImageNet. While CIFAR-10 is the most common benchmark in the adversarial robustness domain, some image transformations work poorly on such small pixelated images. The large and realistic images from Imagenette more closely resemble real-world usage.

All models are pre-trained on ImageNet to speed up training and boost performance. Since RT models are stochastic, we report their average accuracy together with the 95% confidence interval from ten independent runs. Appendix B

| Transform set | | Adv. acc. with different gradient approximations ($\epsilon = 16/255$, 40 steps) | | | |
|---|---|---|---|---|---|
| | Clean | Exact | BPDA | Identity | Combo |
| BaRT full | $88.10 \pm 0.16$ | n/a | $52.32 \pm 0.22$ | $36.49 \pm 0.25$ | $\mathbf{25.24 \pm 0.16}$ |
| BaRT diff. | $87.43 \pm 0.28$ | $\mathbf{26.06 \pm 0.21}$ | $65.28 \pm 0.25$ | $41.25 \pm 0.26$ | n/a |

Table 1: Comparison of attacks using different gradient approximations. Exact uses the exact gradient, BPDA uses the BPDA gradient for most transforms and the identity for a few, Identity uses the identity gradient, and Combo uses exact gradient for differentiable transforms and BPDA gradient otherwise. The defense BaRT full uses a nearly complete set of BaRT transforms ($K = 26$), and BaRT diff. uses a subset of differentiable transforms ($K = 21$). The attack uses PGD with EoT and CE loss.

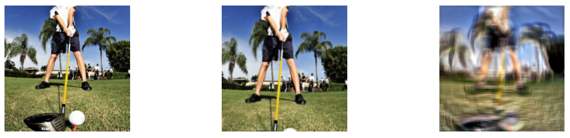

(a) Original (b) Exact crop (c) BPDA crop

Figure 2: Comparison of crop transform output and output of BPDA network trained to approximate crop transform.

has more details on the experiments (network architecture, hyperparameters, etc.).

### 4.4. BPDA is Not Sufficiently Strong

We re-implemented and trained a BaRT model on these datasets, and then evaluated the effectiveness of BPDA attacks against this model. First, we evaluate a full BaRT model (which uses both differentiable and non-differentiable transforms), comparing an attack that uses a BPDA approximation (as Raff et al. did [36]), vs an attack that uses the exact gradient for differentiable transforms and BPDA for non-differentiable transforms. Empirically, we observe that attacks using BPDA gradient approximations are far weaker than the equivalent attack using exact gradient approximations (Table 1). Similarly, on a variant BaRT model that uses only the subset of differentiable transforms, the BDPA attack is worse than an attack that uses the exact gradient for all transforms. BPDA is surprisingly weaker than even a naive attack which approximates all transform gradients with the identity. There are a few possible explanations for the inability of BPDA to approximate transformation gradients well:

1. As Fig. 2 illustrates, BPDA struggles to approximate some transforms accurately. This might be partly because the architecture Raff et al. [36] used (and we use) to approximate each transform has limited functional expressivity: it consists of five convolutional layers with 5x5 kernel and one with 3x3 kernel (all strides are 1), so a single output pixel can only depend on the input pixels fewer than 11 spaces away in any direction ($5 \cdot \lfloor \frac{5}{2} \rfloor + 1 \cdot \lfloor \frac{3}{2} \rfloor = 11$). Considering the inputs for Imagenette are of size $224 \times 224$, some transforms like "crop" which require moving pixels much longer distances are impossible to approximate with such an architecture.
2. The BPDA network training process for solving Eqn. 9 may only find a sub-optimal solution, yielding a poor approximation of the true transformation.
3. During the attack, the trained BPDA networks are given partially transformed images, yet the BPDA networks are only trained with untransformed inputs.
4. Since we are backpropagating through several transforms, one poor transform gradient approximation could ruin the overall gradient approximation.

Appendix B.1 has more details on these experiments. These results show that BaRT's evaluation was overly optimistic and BaRT is not as robust as previously thought.

## 5. Stronger PGD Attacks on RT

### 5.1. Effect of the Permutation of the Transformations

As described in Section 3.1, when computing $g_n$, we randomly sample $n$ independent sequences of transformations (random types and random order). However, our experiments suggest that the attack performs best when the order of transforms is the same for all $n$ images within a batch gradient computation (i.e., a PGD step), and only the boolean and the strength are randomly sampled differently for each image. Table 3 compares these methods on two different attacks, Linear+MB and Linear+Adam, which will be explained in later parts of this section, showing that fixing the order within each batch performs better than sampling it for each image or fixing it across all batches.

The "fixed per PGD step" sampling may seem counter-intuitive as it introduces a biased sampling of $\theta$ and so yields a biased estimator of the gradients. We hypothesize that it is beneficial to the attack because it yields gradients that have smaller variance. Later, we verify this hypothesis and discuss it in more detail in Section 5.5 (Fig. 3). We use this sampling method for the rest of the paper.

### 5.2. Effect of the Loss Function

We experiment with two loss functions that are commonly used for generating adversarial examples: cross entropy loss

| Attacks | Adv. acc. with varying attack steps ($n = 10$) | | | Adv. acc. with varying $n$ (attack steps = 200) | | |
|---|---|---|---|---|---|---|
| | 50 | 200 | 800 | 5 | 10 | 20 |
| CE (EoT) | $82.81 \pm 0.43$ | $72.86 \pm 0.37$ | $65.44 \pm 0.39$ | $75.00 \pm 0.47$ | $72.96 \pm 0.55$ | $71.30 \pm 0.29$ |
| CE (softmax) | $82.37 \pm 0.39$ | $71.05 \pm 0.36$ | $65.06 \pm 0.39$ | $73.82 \pm 0.35$ | $70.71 \pm 0.53$ | $68.51 \pm 0.33$ |
| CE (logits) | $80.79 \pm 0.29$ | $66.68 \pm 0.38$ | $57.72 \pm 0.55$ | $69.94 \pm 0.66$ | $66.65 \pm 0.46$ | $62.68 \pm 0.36$ |
| Linear | $80.67 \pm 0.50$ | $66.11 \pm 0.58$ | $58.26 \pm 0.62$ | $70.67 \pm 0.41$ | $66.59 \pm 0.57$ | $62.48 \pm 0.41$ |
| Linear+MB | $78.51 \pm 0.45$ | $72.66 \pm 0.50$ | $65.28 \pm 0.41$ | $72.47 \pm 0.39$ | $72.51 \pm 0.55$ | $71.06 \pm 0.32$ |
| Linear+LinBP | $82.90 \pm 0.50$ | $70.57 \pm 0.32$ | $65.15 \pm 0.43$ | $75.24 \pm 0.35$ | $72.73 \pm 0.40$ | $70.02 \pm 0.31$ |
| Linear+SGM | $80.10 \pm 0.43$ | $63.75 \pm 0.21$ | $51.68 \pm 0.35$ | $\mathbf{66.93 \pm 0.43}$ | $\mathbf{62.57 \pm 0.31}$ | $59.61 \pm 0.55$ |
| Linear+TG | $80.78 \pm 0.56$ | $68.70 \pm 0.34$ | $59.69 \pm 0.57$ | $71.72 \pm 0.41$ | $67.84 \pm 0.50$ | $65.63 \pm 0.50$ |
| Linear+SGD+Momentum | $75.14 \pm 0.32$ | $73.56 \pm 0.46$ | $73.84 \pm 0.28$ | $76.88 \pm 0.53$ | $73.31 \pm 0.42$ | $70.76 \pm 0.55$ |
| Linear+SGD+Nesterov | $\mathbf{74.95 \pm 0.29}$ | $74.37 \pm 0.38$ | $74.60 \pm 0.39$ | $77.15 \pm 0.49$ | $73.88 \pm 0.54$ | $71.63 \pm 0.41$ |
| Linear+Adam | $80.62 \pm 0.43$ | $62.76 \pm 0.54$ | $46.03 \pm 0.62$ | $67.20 \pm 0.62$ | $63.18 \pm 0.56$ | $58.59 \pm 0.51$ |
| Linear+Adam+SGM | $78.03 \pm 0.53$ | $\mathbf{59.60 \pm 0.49}$ | $\mathbf{38.35 \pm 0.31}$ | $67.14 \pm 0.48$ | $63.03 \pm 0.64$ | $\mathbf{57.72 \pm 0.51}$ |

Table 2: Comparison of different attack techniques on our best RT model. Lower means stronger attack.

| Transform Permutation Method during Attack | Adv. Acc. (Linear+MB) | Adv. Acc. (Linear+Adam) |
|---|---|---|
| All random | $83.07 \pm 0.38$ | $68.23 \pm 0.56$ |
| Fixed per PGD step | $\mathbf{71.05 \pm 0.45}$ | $\mathbf{58.59 \pm 0.51}$ |
| All fixed | $81.45 \pm 0.19$ | $80.17 \pm 0.52$ |

Table 3: Adversarial accuracy of an RT model under attacks using different permutation methods of the transformations for the gradient computation.

(CE) and linear loss (Linear). We consider three variants of the CE loss, based on how we take the expectation over $n$ samples: (1) average of the $n$ losses, (2) loss of the average of the softmax probability, which is proposed by Salman et al. [37], and (3) loss of the average of the $n$ logits. Using the formulation in Section 4.1, $L_n^{eot}$ is equivalent to case (1) of CE loss and $L_n^{logit}$ to case (3) of CE loss. Finally, the linear loss is simply the logit of the true class:

$$L_{\text{linear}}(x, y) := \left[ \underset{\theta \sim p(\theta)}{\mathbb{E}} \left[ f\left( t(x + \delta; \theta) \right) \right] \right]_y \tag{10}$$

Linear or hinge loss is a popular choice of objective function and has been used in several attacks to avoid the vanishing gradient problem caused by the softmax layer [7, 52]. Due to its linearity, taking the average either before or after computing the loss, case (2) and (1) respectively, is equivalent.

The first four rows of Table 2 compare these loss functions. Different attack methods are compared at varying number of PGD steps and RT samples $n$. The widely used EoT method performs the worst of the four. CE loss on average softmax probability performs better than EoT, confirming the observation made by Salman et al. [37]. Linear loss and CE loss on average logits are even better and are consistently the strongest attacks, across multiple choices of $n$ and number of PGD steps. For the rest of this paper, we adopt the linear loss as the main objective function which gets combined with other techniques. In the interest of space, results on the combinations with the other losses are included in Appendix D.

## 5.3. Ensemble Perspective and Transferability

RT can be regarded as an ensemble, with each member using a differently sampled sequence of transforms: the $n$ samples in $g_n(\cdot)$ are equivalent to an ensemble with $n$ random members with partially shared parameters. Since the RT ensemble is stochastic, we may view a white-box attack on RT as a transfer attack from one ensemble to another with very similar structure and some shared parameters.

Given this interpretation, we apply four techniques designed to enhance the transferability of adversarial examples to improve the attack success rate on RT. We tried momentum boosting (MB) [13], modifying backward passes to treat activations as linear (LinBP) [19] or to prefer the gradient through skip connections (SGM) [47], and simply using a targeted attack with the linear loss function (Linear+TG) [52]. Results in Table 2 (5th – 8th rows) show that SGM convincingly outperforms the other three techniques across all settings.

## 5.4. SGD Perspective and Acceleration

The output of RT in Eqn. 2 is an expectation which can be regarded as an infinite sum over the samples of $\theta$. As mentioned in Section 5.1, the Monte Carlo approximation of the expectation in Eqn. 2 gives unbiased but potentially high-variance gradients which undermines the convergence rate of naive SGD. Thus, we hypothesized that the attack could benefit from techniques aimed at speeding up the convergence of SGD. To test this hypothesis, we experiment with Nesterov acceleration on SGD [41] and an optimizer with an adaptive learning rate such as Adam [27].

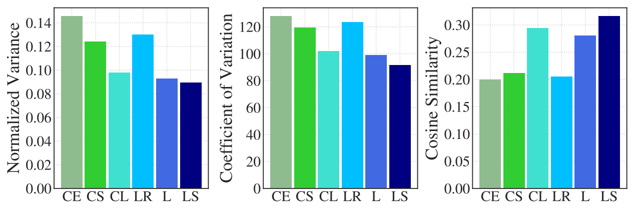


Figure 3: Three metrics comparing the normalized spread of the gradients of different attack objectives. CE: CE (EoT), CS: CE (softmax), CL: CE (logit), LR: Linear with all random perm., L: Linear, LS: Linear+SGM.

Additionally, Yin et al. [49] show that the Adam optimizer also improves the transferability of adversarial examples which may further improve the attack as explained by the fact that RT model acts like an ensemble (Section 5.3). We also find that normalizing the gradients to have a unit $\ell_p$-norm before scaling by the step size, as is done in the normal PGD attack, is crucial to the success of the attacks. Not normalizing the gradients or using the change of variable trick in Carlini and Wagner [7] results in weaker attacks.

The last three rows of Table 2 compare these three optimizers. Adam outperforms SGD both with and without Nesterov acceleration in all cases apart from one.

Finally, we combine all the best techniques found so far, i.e. the linear loss, Adam optimizer, and skip connection backward pass, into one attack, Linear+Adam+SGM. As shown in the last row of Table 2, Linear+Adam+SGM achieves the highest attack success rate of any method, and outperforms all other methods given enough attack steps. Compared to the naive EoT attack with CE loss, Linear+Adam+SGM improves the attack success rate by a large margin of up to 27%.

### 5.5. Attacks' Effectiveness and Variance

In this section, we explain the success of some of the above attack objectives by analyzing the variance of their respective gradients. Classical convergence analysis of SGD on convex functions states that the convergence rate is inversely proportional to the variance of the gradient estimator [34, 17]. Thus, we naturally prefer gradients with smaller variance, even though they may be biased, which is in fact the principle behind many well-known variance reduction techniques [25, 11, 38, 16].

Now consider the estimated objective $L_n^{\text{logit}}$ (Eqn. 3) and a sample of its estimated gradient (indexed by $j$) given by

$$\nabla L_{n,j}^{\text{logit}}(x,y) \coloneqq \nabla_{\mathbf{x}} L\left(\frac{1}{n}\sum_{i=1}^{n} f(t(x;\theta_{j,i})), y\right) \tag{11}$$

Note that Eqn. 11 is a biased estimator of the true gradient [37]. The sample mean and the sample variance (averaged over dimension) of this estimator are then given by

$$\mu_{m,n} \coloneqq \nabla \overline{L}_{m,n} = \frac{1}{m}\sum_{j=1}^{m} \nabla L_{n,j} \tag{12}$$

$$\sigma_{m,n}^2 \coloneqq \frac{1}{d}\frac{1}{m-1}\sum_{j=1}^{m}\left\|\nabla \overline{L}_n - \nabla L_{n,j}\right\|_2^2 \tag{13}$$

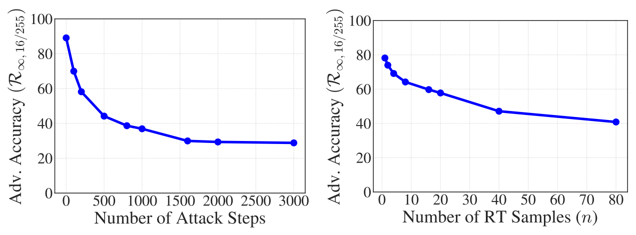
where the dependence on $(x,y)$ is dropped to declutter. For a given loss function, using larger $n$ reduces $\sigma_{m,n}^2$ as expected, but comparing between different loss functions is less straightforward since the norms of the gradients scale differently (top-left plot in Fig. 3). To this end, we compute three other metrics that measure the spread of the gradient samples and are invariant to scaling, namely scale-normalized variance, coefficient of variation (CV), and cosine similarity between each $\nabla L_{n,j}$ and the mean $\mu_{m,n}$. Fig. 3 confirms our hypothesis that small normalized variance (small CV and large cosine similarity) implies a stronger attack. The trends and ranking match exactly with those of the adversarial accuracy shown in Table 2 and Table 3. We also observe that the estimated gradients are extremely noisy which might be the key limitation of the naive first-order attack. In Appendix D.1, we describe how we compute these metrics and present more plots with different values of $n$.

This observation suggests that optimization techniques that reduce gradient variances enable stronger attacks against randomized defenses. However, most variance-reduced SGD methods are specific to the finite-sum problem [25, 11, 38, 16], whereas our objective is an infinite sum. Some techniques may be adapted to our problem, but we leave this direction to future works.

## 6. Hyperparameter Tuning

One major challenge in implementing RT is selecting the defense hyperparameters which include the $K$ transformation types, the number of transformations to apply ($S$), and their parameters ($\alpha$ and $p$). To improve the robustness of RT defense, we use Bayesian optimization (BO), a well-known black-box optimization technique, to fine-tune the two real-valued parameters, $\alpha$ and $p$ [40]. In this case, BO models the hyperparameter tuning as a Gaussian process where the objective function takes in $\alpha$ and $p$, trains a neural network for RT, and outputs adversarial accuracy under some pre-defined $\ell_\infty$-budget $\epsilon$ as the metric used for optimization.

Since BO quickly becomes ineffective as we increase the dimensions of the search space, we choose to tune either $\alpha$ or $p$, never both, for each of the $K$ transformation types. For transformations that have a tunable $\alpha$, we fix $p = 1$ (e.g., noise injection, affine transform). For the transformations without an adjustable strength $\alpha$, we only tune $p$ (e.g., Laplacian filter, horizontal flip). Additionally, because BO does not natively support categorical or integral variables, we ex-



(a) Varying attack steps. (b) Varying $n$

Figure 4: Adversarial accuracy of the final RT model under a wide range of (a) attack steps and (b) $n$.

periment with different choices for $K$ and $S$ without the use of BO. Therefore, our BO problem must optimize over $K$ (up to 33) variables, far more than are typically present when doing model hyperparamter tuning using BO.

Mathematically, the objective function $\psi$ is defined as

$$\psi : [0,1]^K \to \mathcal{R}_{\infty,\epsilon} \in [0,1] \tag{14}$$

where the input is $K$ real numbers between 0 and 1, and $\mathcal{R}_{\infty,\epsilon}$ denotes the adversarial accuracy or the accuracy on $x_{\text{adv}}$ as defined in Eqn. 1. Since $\psi$ is very expensive to evaluate as it involves training and testing a large neural network, we employ the following strategies to reduce the computation: (1) only a subset of the training and validation set is used, (2) the network is trained for fewer epochs with a cosine annealing learning rate schedule to speed up convergence [32], and (3) the attack used for computing $\mathcal{R}_{\infty,\epsilon}$ is weaker but faster. Even with these speedups, one BO run still takes approximately two days to complete on two GPUs (Nvidia GeForce GTX 1080 Ti). We also experimented with other sophisticated hyperparameter-tuning algorithms based on Gaussian processes [4, 26, 14] but do not find them more effective. We summarize the main steps for tuning and training an RT defense in Algorithm 1.

## 7. Evaluations on RT Defense

### 7.1. Adversarial Robustness of RT Defense

**Finding best RT model.** BO generally finds a good configuration of hyperparameters for RT, but the optimality deteriorates as the number of variables or tuned hyperparameters increases. To mitigate this problem, we come up with the following procedure. We run multiple BO experiments (Algorithm 1) on different subsets of transformation types to identify which transformations are most/least effective in order to reduce $K$ as well as the number of hyperparameters our final run of BO has to tune. We then repeat Algorithm 1 initialized with the input-output pairs from the prior runs of BO to obtain a new set of hyperparameters. Finally, we remove the transformations whose $p$ or $\alpha$ has been set to zero by the first run of BO, and we run BO once more with this filtered subset of transformations. At the end of this expensive procedure, we obtain the best and final RT model that we use in the experiments throughout this paper. This model only has 18 transformation types out of the original 33. More details on this final model are provided in Appendix C.1.

This RT model is evaluated against the best attack (Linear+Adam+SGM) under a large number of steps and samples $n$ (Fig. 4). As expected, the adversarial accuracy drops as we increase the number of PGD steps and $n$ until it plateaus after around 1600 steps. The adversarial accuracy at 3000 steps is reported in Table 4 along with the accuracy of a normal network and other well-known defenses against adversarial robustness including adversarial training (AT) [33] and TRADES [50]. All models are pre-trained on Ima-

**Algorithm 1:** Tuning and training RT defense.

```
   Input: Set of transformation types, $n$, $p$, $\epsilon$
   Output: $g^*(\cdot)$, $\mathcal{R}$, $\mathcal{R}_{p,\epsilon}$
   Data: Training data $(\boldsymbol{X}^{\text{train}}, \boldsymbol{Y}^{\text{train}})$, test data
         $(\boldsymbol{X}^{\text{test}}, \boldsymbol{Y}^{\text{test}})$
   // Starting Bayesian optimization (BO)
1  Sub-sample $(\boldsymbol{X}^{\text{train}}, \boldsymbol{Y}^{\text{train}})$ and split it into BO's
     training data $(\boldsymbol{X}^{\text{train}}_{\text{BO}}, \boldsymbol{Y}^{\text{train}}_{\text{BO}})$ and validation data
     $(\boldsymbol{X}^{\text{val}}_{\text{BO}}, \boldsymbol{Y}^{\text{val}}_{\text{BO}})$.
2  $\mathcal{R}^*_{p,\epsilon} \leftarrow 0$          // Best adversarial accuracy
3  $\{(p^*_i, \alpha^*_i)\}_{i=1}^K \leftarrow 0$     // Best RT hyperparameters
4  for step $\leftarrow 0$ to MAX_BO_STEPS do
       // Running one trial of BO
5      BO specifies $\{(p_i, \alpha_i)\}_{i=1}^K$ to evaluate.
6      Train an RT model on $(\boldsymbol{X}^{\text{train}}_{\text{BO}}, \boldsymbol{Y}^{\text{train}}_{\text{BO}})$ with
         hyperparameters $\{(p_i, \alpha_i)\}_{i=1}^K$ to obtain $g$.
7      Test $g$ by computing $\mathcal{R}_{p,\epsilon}$ on $(\boldsymbol{X}^{\text{val}}_{\text{BO}}, \boldsymbol{Y}^{\text{val}}_{\text{BO}})$ using a
         weak but fast attack.
8      if $\mathcal{R}_{p,\epsilon} > \mathcal{R}^*_{p,\epsilon}$ then
9          $\mathcal{R}^*_{p,\epsilon} \leftarrow \mathcal{R}_{p,\epsilon}$
10         $\{(p^*_i, \alpha^*_i)\}_{i=1}^K \leftarrow \{(p_i, \alpha_i)\}_{i=1}^K$
11     else if No improvement for some steps then
12         break;
   // Full training of RT
13 Train an RT model on $(\boldsymbol{X}^{\text{train}}, \boldsymbol{Y}^{\text{train}})$ with best
     hyperparameters $\{(p^*_i, \alpha^*_i)\}_{i=1}^K$ to obtain $g^*$.
14 Evaluate $g^*$ by computing $\mathcal{R}$ and $\mathcal{R}_{p,\epsilon}$ on $(\boldsymbol{X}^{\text{test}}, \boldsymbol{Y}^{\text{test}})$
     using a strong attack.
```

| Models | Clean Acc. | Adv. Acc. |
|---|---|---|
| Normal | **95.40** | 0.00 |
| AT | 78.20 | **37.10** |
| TRADES | 78.60 | 20.50 |
| RT | 89.08 | 28.82 |

Table 4: Clean and adversarial accuracy at $\epsilon = 16/255$ for multiple models. All models apart from RT is evaluated by AutoAttack [9], a strong ensemble of four attacks.

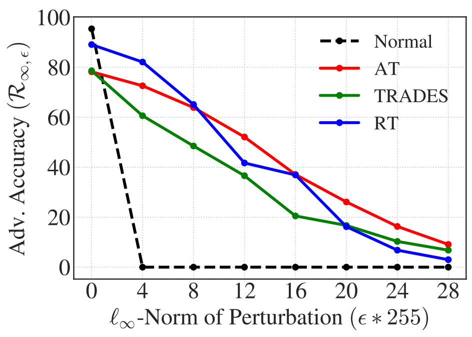


Figure 5: Adversarial accuracy of multiple defenses under a range of $\ell_\infty$-norm of the perturbation.

| Omitted Transformations | Clean Acc. | Adv. Acc. |
|---|---|---|
| Noise injection | $92.27 \pm 0.16$ | $31.96 \pm 0.54$ |
| Blur filter | $89.04 \pm 0.34$ | $38.43 \pm 0.68$ |
| Color space | $90.65 \pm 0.27$ | $47.08 \pm 0.53$ |
| Edge detection | $87.12 \pm 0.45$ | $15.58 \pm 0.22$ |
| Lossy compression | $88.19 \pm 0.19$ | $16.14 \pm 0.31$ |
| Geometric transforms | $86.87 \pm 0.57$ | $43.68 \pm 0.56$ |
| Stylization | $\mathbf{85.71 \pm 0.52}$ | $\mathbf{13.26 \pm 0.34}$ |

Table 5: RT's performance when each of the transformation groups is removed.

geNet similarly to RT, and they are trained and tested using $\epsilon = 16/255$. Fig. 5 further compares the robustness given varying $\ell_\infty$-norm $\epsilon$ budgets for the perturbation. Here, we use Linear+Adam+SGM with 1000 steps to attack RT. It is clear that our best RT model is still less robust than AT at most of the perturbation sizes and performs better than AT only on clean samples and on adversarial examples with small $\epsilon$.

## 7.2. Ablation Studies

**Effect of the Number of Transformations.** Fig. 12 (Appendix E.3) shows that more transformations (larger $S$) increase robustness but lower accuracy on benign samples. The improvement on the robustness plateaus after $S = 16$, but the clean accuracy keeps decreasing, suggesting that after a certain point, there is no benefit for increasing $S$.

**Effect of Transformation Groups.** Choosing the best set of transformation types to use is a computationally expensive problem. There are many more transformations that can be applied outside of the 33 types we choose, and the number of possible combinations grows exponentially. BO gives us an approximate solution but is by no means perfect. Here, we take a step further to understand the importance of each transformation group. Table 5 shows performance of RT models obtained from BO after removing one group of transformations at a time. Without stylization, edge detection, and lossy compression, RT performs significantly worse than

| Choice of Randomness in RT | Clean Acc. | Adv Acc. |
|---|---|---|
| Random perm. & param | $89.08 \pm 0.32$ | $\mathbf{58.59 \pm 0.51}$ |
| Fixed perm. & random param | $87.93 \pm 0.44$ | $56.80 \pm 0.40$ |
| Fixed perm. & fixed param | $88.40$ | $14.70$ |
| No transformation | $\mathbf{93.20}$ | $0.00$ |

Table 6: Effects of randomness on RT. Adv. acc. is computed with the Linear+Adam+SGM attack (200 steps, $n = 20$).

when all transformation groups are present, implying that they play a crucial role in offering robustness. We also conduct more experiments attempting to quantify the relative importance of the transformation groups in Appendix E.2.

**Does randomness matter?** Table 6 compares the same RT model under four scenarios. The first row is the normal scenario where the transformations are randomly permuted and their parameters (boolean and strength) are randomly sampled. When the permutation is fixed, the adversary always sees the same set of transformations with either random (2nd row) or fixed parameters (3rd row). Table 6 shows that only fixing the permutation barely affects the accuracy and the robustness, while fixing both significantly drops the robustness. In the fourth row, where no transformation is applied, the model has the highest clean accuracy as it is given a clean non-corrupted input, but is not at all robust. This indicates that randomness, as expected, clearly contributes to the robustness of RT.

**Combining RT with Adversarial Training** Using the same hyperparameters as the best RT model, we adversarially train an RT +AT model using 10-step PGD with the linear loss ($n = 4$). Since the use of RT makes the training process approximately four times longer than normal AT or 40 times longer than normal training, we can only afford this relatively weak attack. This attack is so weak that it is only able to reduce the accuracy by $2 - 3\%$ and therefore fails to noticeably boost model robustness. The final model does not end up generalizing well to adversarial examples not seen during training. Compared to the best RT model, the RT+AT model performs slightly better on PGD with linear loss but slightly worse on the strongest attack.

## 8. Conclusion

While recent papers report state-of-the-art robustness with RT defenses, our evaluations show that RT generally under-performs existing defenses like AT when met with a stronger attack, even after fine-tuning the hyperparameters of the defense. Through our experiments, we found that non-differentiability and high-variance gradients can seriously inhibit adversarial optimization, so we recommend using exact gradients and variance reduction techniques where possible in the evaluation of future RT defenses.

## References

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. 1, 3

[2] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 284–293, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. 3

[3] Christopher Bender, Yang Li, Yifeng Shi, Michael K. Reiter, and Junier Oliva. Defense through diverse directions. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 756–766. PMLR, July 2020. 1, 2

[4] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages I–115–I–123, Atlanta, GA, USA, 2013. JMLR.org. 7

[5] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 387–402, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 2

[6] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599 [cs]*, Dec. 2010. 14

[7] Nicholas Carlini and D. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017. 5, 6

[8] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR, June 2019. 1, 2

[9] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2206–2216. PMLR, July 2020. 7

[10] Ashok Cutkosky and Francesco Orabona. Momentum-based variance reduction in non-convex SGD. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 15

[11] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. 6

[12] Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018. 1

[13] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Mar. 2018. 5, 15

[14] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. 7

[15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. 2

[16] Eduard Gorbunov, Filip Hanzely, and Peter Richtarik. A unified theory of SGD: Variance reduction, sampling, quantization and coordinate descent. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 680–690. PMLR, Aug. 2020. 6

[17] Robert M Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. SGD: General analysis and improved rates. In *International Conference on Machine Learning*, Los Angeles, United States, June 2019. 6

[18] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. 1

[19] Yiwen Guo, Qizhang Li, and Hao Chen. Backpropagating linearly improves transferability of adversarial examples. In *NeurIPS*, 2020. 5

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 13

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 13

[22] Warren He, Bo Li, and Dawn Song. Decision boundary analysis of adversarial examples. In *International Conference on Learning Representations*, 2018. 1

[23] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural

network robustness against adversarial attack. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–597, 2019. 1, 2

[24] Jeremy Howard. Fastai/imagenette. fast.ai, Mar. 2021. 3

[25] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. 6

[26] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabas Poczos, and Eric P. Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020. 7

[27] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014. 5

[28] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672, 2019. 1

[29] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018. 14

[30] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *ECCV (7)*, pages 381–397, 2018. 1, 2

[31] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-BNN: Improved adversarial defense through robust bayesian neural network. In *International Conference on Learning Representations*, 2019. 1, 2

[32] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. 7, 13

[33] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 1, 7

[34] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009. 6

[35] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python. 2014. 14

[36] Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. Barrage of random transforms for adversarially robust defense. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6521–6530, Long Beach, CA, USA, June 2019. IEEE. 1, 2, 3, 4, 12, 13

[37] Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 3, 5, 6

[38] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 2013. 6

[39] Richard Shin and Dawn Song. JPEG-resistant adversarial images. In *Machine Learning and Computer Security Workshop (Co-Located with NeurIPS 2017)*, Long Beach, CA, USA, 2017. 3, 13

[40] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 6, 14

[41] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147, Atlanta, GA, USA, 2013. JMLR.org. 5

[42] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. 2

[43] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1633–1645. Curran Associates, Inc., 2020. 1

[44] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018. 18

[45] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019. 17

[46] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020. 18

[47] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with ResNets. In *International Conference on Learning Representations*, 2020. 5

[48] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018. 1, 2

[49] Heng Yin, Hengwei Zhang, Jindong Wang, and Ruiyu Dou. Improving the transferability of adversarial examples with the adam optimizer. *arXiv:2012.00567 [cs]*, Dec. 2020. 6

[50] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*, 2019. 7

[51] Yuchen Zhang and Percy Liang. Defending against whitebox adversarial attacks via randomized discretization. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 684–693. PMLR, Apr. 2019. 1, 2

[52] Zhengyu Zhao, Zhuoran Liu, and Martha Larson. On success and simplicity: A second look at transferable targeted attacks. *arXiv:2012.11207 [cs]*, Feb. 2021. 5

## A. Details on the Image Transformations

The exact implementation of RT models and all the transformations will be released. Here, we provide some details on each of the transformation types and groups. Then, we describe how we approximate some non-differentiable functions with differentiable ones.

### A.1. List of Transformation Types

#### Noise injection

- **Erase:** Set the pixels in a box with random size and location to zero.
- **Gaussian noise:** Add Gaussian noise to each pixel.
- **Pepper:** Zero out pixels with some probability.
- **Poisson noise:** Add Poisson noise to each pixel.
- **Salt:** Set pixels to one with some probability.
- **Speckle noise:** Add speckle noise to each pixel.
- **Uniform noise:** Add uniform noise to each pixel.

#### Blur filtering

- **Box blur:** Blur with randomly sized mean filter.
- **Gaussian blur:** Blur with randomly sized Gaussian filter with randomly chosen variance.
- **Median blur:** Blur with randomly sized median filter.
- **Motion blur:** Blur with kernel for random motion angle and direction.

#### Color-space alteration

- **HSV:** Convert to HSV color-space, add uniform noise, then convert back.
- **LAB:** Convert to LAB color-space, add uniform noise, then convert back.
- **Gray scale mix:** Mix channels with random proportions.
- **Gray scale partial mix:** Mix channels with random proportions, then mix gray image with each channel with random proportions.
- **Two channel gray scale mix:** Mix two random channels with random proportions.
- **One channel partial gray:** Mix two random channels with random proportions, then mix gray image with other channel.
- **XYZ:** Convert to XYZ color-space, add uniform noise, then convert back.
- **YUV:** Convert to YUV color-space, add uniform noise, then convert back.

#### Edge detection

- **Laplacian:** Apply Laplacian filter.
- **Sobel:** Apply the Sobel operator.

#### Lossy compression

- **JPEG compression:** Compress image using JPEG to a random quality.
- **Color precision reduction:** Reduce color precision to a random number of bins.
- **FFT perturbation:** Perform FFT on image and remove each component with some probability.

#### Geometric transforms

- **Affine:** Perform random affine transformation on image.
- **Crop:** Crop image randomly and resize to original shape.
- **Horizontal flip:** Flip image across the vertical.
- **Swirl:** Swirl the pixels of an image with random radius and strength.
- **Vertical flip:** Flip image across the horizontal.

#### Stylization

- **Color jitter:** Randomly alter the brightness, contrast, and saturation.
- **Gamma:** Randomly alter gamma.
- **Sharpen:** Apply sharpness filter with random strength.
- **Solarize:** Solarize the image.

#### Non-differentiable (for BPDA Tests Only)

- **Adaptive histogram:** Equalize histogram in patches of random kernel size.
- **Chambolle denoise:** Apply Chambolle's total variation denoising algorithm with random weight (can be implemented differentiably but was not due to time constraints).
- **Contrast stretching:** Pick a random minimum and maximum pixel value to rescale intensities (can be implemented differentiably but was not due to time constraints).
- **Histogram:** Equalize histogram using a random number of bins.

#### Unused transforms from BaRT

- **Seam carving:** Algorithm used in Raff et al. [36] has been patented and is no longer available for open-source use.
- **Wavelet denoising:** The implementation in Raff et al. [36] is incomplete.
- **Salt & pepper:** We have already used salt and pepper noise separately.
- **Non-local means denoising:** The implementation of NL means denoising in Raff et al. [36] is too slow.

## A.2. Differentiable Approximation

Some of the transforms contain non-differentiable operations which can be easily approximated with differentiable functions. Specifically, we approximate the rounding function in JPEG compression and color precision reduction, and the modulo operator in all transformations that require conversion between RGB and HSV color-spaces (HSV alteration and color jitter). Note that we are not using the non-differentiable transform on the forward pass and a differentiable approximation on the backward pass (like in BPDA). Instead, we are using the differentiable version both when performing the forward pass and when computing the gradient.

We take the approximation of the rounding function from Shin and Song [39] shown in Eqn. 15.

$$\lfloor x \rceil_{\text{approx}} = \lfloor x \rceil + (x - \lfloor x \rceil)^3 \tag{15}$$

For the modulo or the remainder function, we approximate it using the above differentiable rounding function as a basis.

$$\text{mod}(x) = \begin{cases} x - \lfloor x \rceil & \text{if } x > \lfloor x \rceil \\ x - \lfloor x \rceil + 1 & \text{otherwise} \end{cases} \tag{16}$$

To obtain a differentiable approximation, we can replace the rounding operator with its smooth version in Eqn. 15. This function (approximately) returns decimal numbers or a fractional part of a given real number, and it can be scaled to approximate a modulo operator with any divisor.

Note that these operators are step functions and are differentiable almost everywhere, like ReLU. However, their derivatives are always zero (unlike ReLU), and so a first-order optimization algorithm would still fail on these functions.

## B. Detailed Experimental Setup

All of the experiments are evaluated on 1000 randomly chosen test samples. Since we choose the default $n$ to be 40 for inference and 20 for the attacks, the experiments are at least 20 times more expensive than usual, and we cannot afford enough computation to run a large number of experiments on the entire test set.

The networks used in this paper are ResNet-34 [20] for Imagenette and Pre-activation ResNet-18 [21] for CIFAR-10. In all of the experiments, we use a learning rate of 0.01, batch size of 128, and weight decay of 0.0005. We use cosine annealing schedule [32] for the learning rate with a period of 20 epochs which also doubles after every period. All models are trained for 140 epochs, and we save the weights with the highest accuracy on the held-out validation data (which does not overlap with the training or test set).

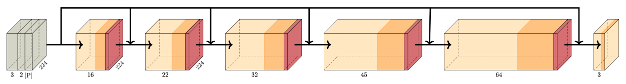


Figure 6: Fully-convolutional BPDA network from Raff et al. [36]. The network has six convolutional layers. All layers have a stride of 1. The first five layers have kernel size of 5 and padding size of 2, and the last layer has a kernel size of 3 and padding size of 1. The input consists of more than 5 channels, 3 of which are for the image RGB channels, 2 of which are CoordConv channels that include the coordinates of each pixel at that pixel's location, and the remaining channels are the parameters for the transformation copied at each pixel location. The network contains a skip connection from the input to each layer except the final layer.

### B.1. Details on BPDA Experiments

We used the following setup for the differentiability related experiments conducted in Section 4.4:

- Each accuracy is an average over 10 trials on the same set of 1000 Imagenette images.
- The defense samples $S = 10$ transforms from the full set of $K$ transforms.
- The image classifier uses a ResNet-50 architecture like in Raff et al. [36] trained on transformed images for 30 epochs.
- The attack uses 40 PGD steps of size $4/255$ with an $\epsilon = 16/255$ to minimize the EoT objective.

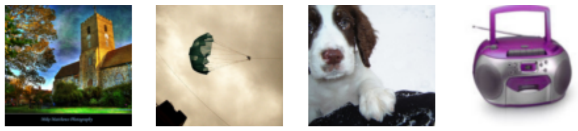
The BPDA network architecture is the same used by Raff et al. [36] and is outlined in Figure 6. Here are more details on BPDA training:

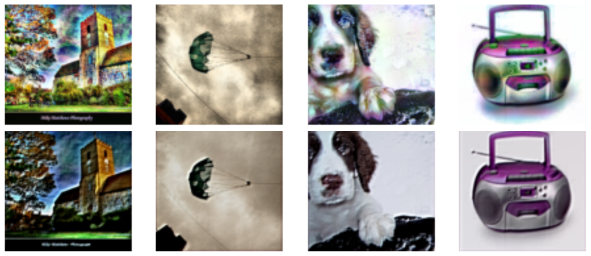
- All BPDA networks were trained using Adam with a learning rate of 0.01 for 10 epochs.
- All networks achieve a per-pixel MSE below 0.01. The outputs of the BPDA networks are compared to the true transform outputs for several different transform types in Figure 7.

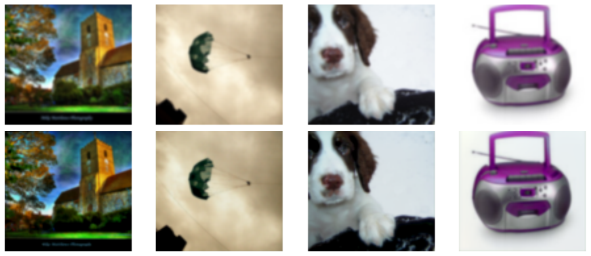
The specific set of transforms used in each defense are the following:

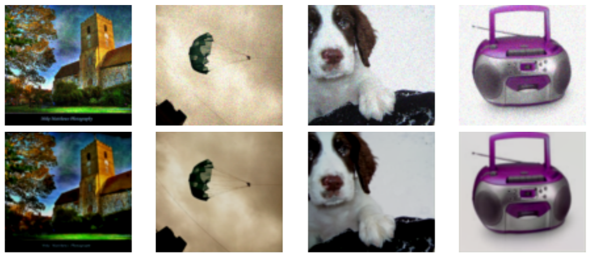
- **BaRT all:** adaptive histogram, histogram, bilateral blur, box blur, Gaussian blur, median blur, contrast stretching, FFT, gray scale mix, gray scale partial mix, two channel gray scale mix, one channel gray scale mix, HSV, LAB, XYZ, YUV, JPEG compression, Gaussian noise, Poisson noise, salt, pepper, color precision reduction, swirl, Chambolle denoising, crop.

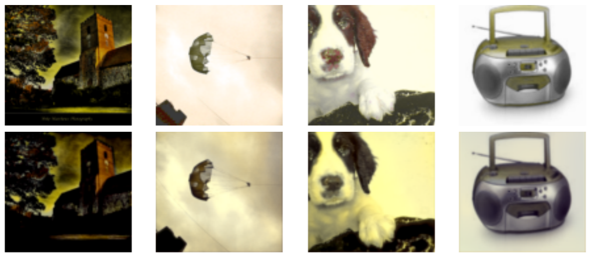
- **BaRT diff:** all of the BaRT all transforms excluding adaptive histogram, histogram, contrast stretching, and Chambolle denoising.

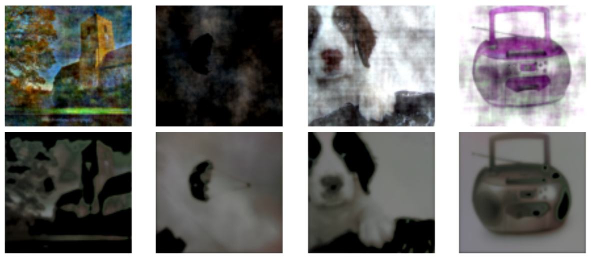

(a) Original


(b) Adaptive histogram


(c) Box blur


(d) Poisson noise


(e) HSV color alteration


(f) FFT

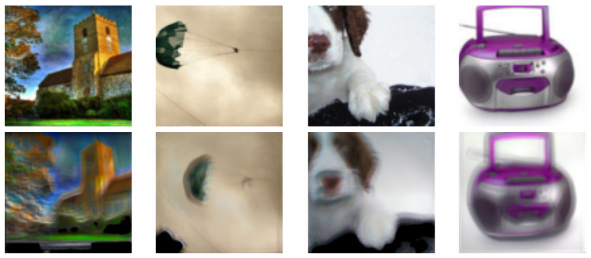

(g) Crop

Figure 7: Comparison of the true transformed outputs (top row) and outputs of respective BPDA networks (bottom row) for six different transformation types.

## C. Details on Bayesian Optimization

We use the Ray Tune library for RT's hyperparameter tuning in Python [29]. The Bayesian optimization tool is implemented by Nogueira [35], following analyses and instructions by Snoek et al. [40] and Brochu et al. [6]. As mentioned in Section 6, we sub-sample the data to reduce computation for each BO trial. Specifically, we use 10% and 20% of the training samples for Imagenette and CIFAR-10 respectively (Algorithm 1, line 1). The models are trained with the same transformations and hyperparameters used during inference, and here, $n$ is set to one during training, just as is done during standard data augmentation. We use 200 samples to evaluate each BO run in line 7 of Algorithm 1, and the attack we use in this step (the "weak but cheap" attack) is still our best version (Linear+Adam+SGM) but with only 100 steps and $n = 10$.

One BO experiment executes two BO's in parallel. The maximum number of BO runs is 160, but we terminate the experiment if no improvement has been made in the last 40 runs after a minimum of 80 runs have taken place. The runtime depends on $S$ and the transformation types used. In our typical case, when all 33 transformation types are used and $S = 14$, one BO run takes almost an hour on an Nvidia GeForce GTX 1080 Ti for Imagenette. One BO experiment then takes about two days to finish.

| Attacks | Adv. acc. with varying attack steps ($n = 10$) | | | Adv. acc. with varying $n$ (attack steps = 200) | | |
|---|---|---|---|---|---|---|
| | 50 | 200 | 800 | 5 | 10 | 20 |
| CE+MB (softmax) | $80.16 \pm 0.34$ | $74.11 \pm 0.53$ | $67.19 \pm 0.62$ | $73.38 \pm 0.52$ | $74.18 \pm 0.68$ | $74.05 \pm 0.27$ |
| CE+MB+VR (softmax) | $80.32 \pm 0.26$ | $74.06 \pm 0.47$ | $66.97 \pm 0.38$ | $73.60 \pm 0.42$ | $74.76 \pm 0.61$ | $74.40 \pm 0.47$ |
| Linear+MB+VR | $79.31 \pm 0.39$ | $73.01 \pm 0.54$ | $64.60 \pm 0.57$ | $72.41 \pm 0.60$ | $72.13 \pm 0.42$ | $70.91 \pm 0.51$ |

Table 7: Comparison of the adversarial accuracy ($\epsilon = 16/255$) when using additional attack methods that we did not include in Table 2 on our best RT model. We evaluate the attacks at varying number of attack iterations (first three columns) and varying $n$ (last three columns). Lower means stronger attack.

In line 13 and 14 of Algorithm 1, we now use the full training set and 1000 test samples as mentioned earlier. During the full training, $n$ is set to four which increases the training time by approximately four times. We find that using a larger $n$ is beneficial to both the clean and the adversarial accuracy, but $n$ larger than four does not make any significant difference.

## C.1. Details on the Final RT Model

We have described the procedure which we used to obtain our best RT model in Section 7.1. The final set of 18 transformation types used in this model are color jitter, erase, gamma, affine, horizontal flip, vertical flip, Laplacian filter, Sobel filter, Gaussian blur, median blur, motion blur, Poisson noise, FFT, JPEG compression, color precision reduction, salt noise, sharpen, and solarize. $S$ is set to 14.

## D. Additional Experiments on the Attacks

Table 7 includes a few more attack techniques that we did not include in Table 2. "VR" denotes our experimental variance reduction technique inspired by Cutkosky and Orabona [10]. We adapted this technique to work with momentum boosting (MB) [13], but due to computational constraints, we make an approximation by saving the previous gradient instead of recomputing a new one with a new set of random samples. This technique boosts the performance of MB alone slightly but is still far inferior to our best attack.

We further compare the baseline attack on RT models to our best attack at different numbers of steps in Fig. 8. The baseline (CE loss with EoT and fully random transform permutation) only reduces the accuracy to around 71% which plateaus after approximately 1000 steps. On the other hand, our best attack (Linear+Adam+SGM), which is used throughout the paper, achieves adversarial accuracy of 28% at 3000 steps, improving from the baseline by 43%. This is an enormous gap that indicates inadequate evaluations and a significant overestimate of the robustness in previous works.

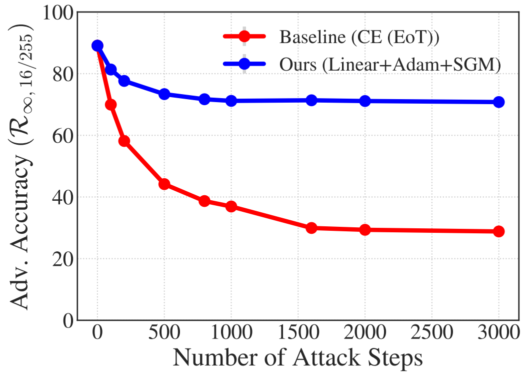


Figure 8: Adversarial accuracy achieved by the baseline attack (CE loss with EoT and fully random transform permutation) and our best attack (Linear+Adam+SGM) under $\epsilon = 16/255, n = 20$ on Imagenette dataset.

## D.1. Variance of Gradients

We have described how we compute the sample variance of the gradients in Section 5.5. Here, we provide detailed calculations of the other three metrics using the same notation from Eqn. 12 and Eqn. 13. First, the cosine similarity is computed between the mean gradient and all $m$ samples and then averaged.

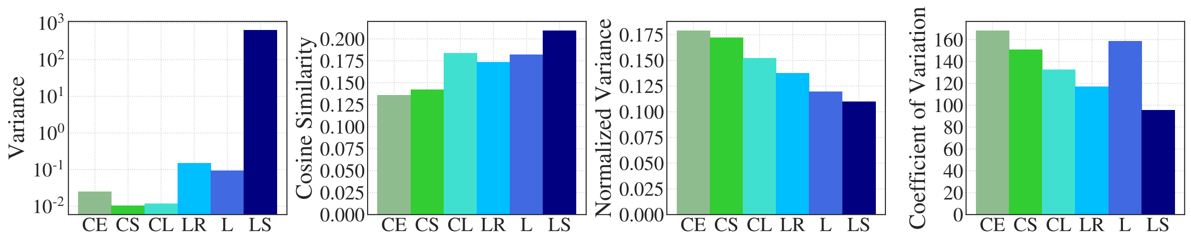
$$\cos_{m,n}(L) := \frac{1}{m} \sum_{j=1}^{m} \frac{\langle \nabla L_{n,j}, \mu_{m,n} \rangle}{\|\nabla L_{n,j}\|_2 \cdot \|\mu_{m,n}\|_2} \tag{17}$$

The scaled-normalized variance is simply the normal variance divided by the squared $\ell_2$-norm of the mean gradient:

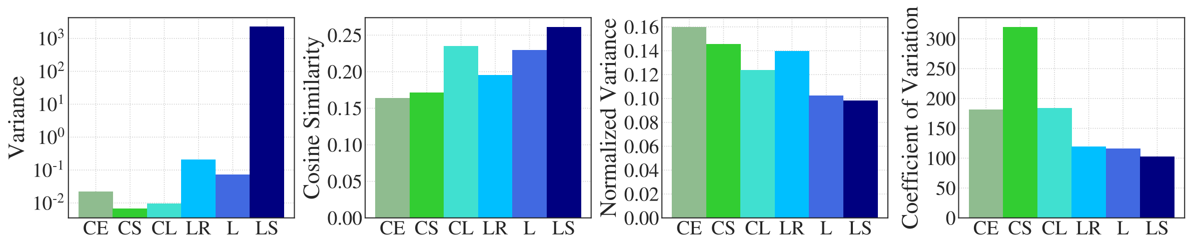
$$\sigma^2_{\text{norm-}m,n} := \frac{\sigma^2_{m,n}}{\|\mu_{m,n}\|_2^2} \tag{18}$$

and lastly, the coefficient of variation (CV), which is a normalized measure of spread based on standard deviation, is given as an average of the element-wise standard deviations divided by the mean:

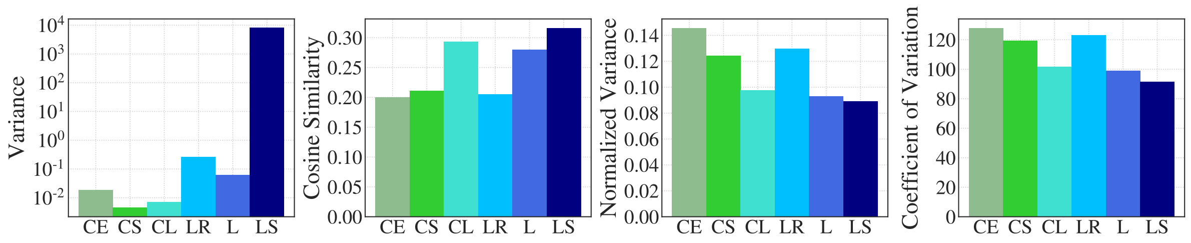
$$\text{CV}_{m,n} := \frac{1}{d} \sum_{i=1}^{d} \frac{\left( \frac{1}{m-1} \sum_{j=1}^{m} \left( [\nabla L_{n,j}]_i - [\mu_{m,n}]_i \right)^2 \right)^{\frac{1}{2}}}{|[\mu_{m,n}]_i|} \tag{19}$$



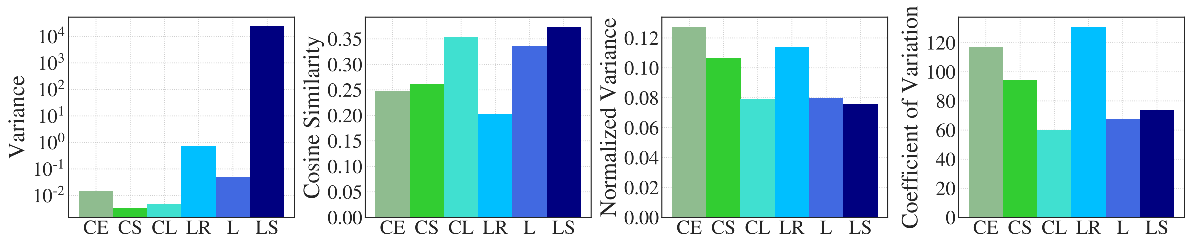
(a) $n = 4$



(b) $n = 8$



(c) $n = 16$



(d) $n = 32$

Figure 9: Comparisons of six different attack objectives on four measurements of the spread of their gradients. The gradients are also computed with different numbers of Monte Carlo samples $n = 4, 8, 16, 32$ corresponding to (a),(b),(c), and (d) respectively. CE: CE loss (EoT), CS: CE loss (softmax), CL: CE loss (logit), LR: Linear loss with all random transform permutations, L: Linear loss, LS: Linear loss combined with SGM. The gradients of RT models are generally very noisy and can vary in multiple orders of magnitude. To compute meaningful metrics, we trim 10% of the gradients with the largest and the smallest norm.

where $[v]_i$ denotes the $i$-th index of the vector $v$, and $d$ is the number of input dimensions.

Fig. 9 includes all four metrics at different values of $n$. Roughly speaking, we can sort five out of the six given attacks from strongest to weakest in this order: LS $>$ L $\approx$ CL $>$ CS $>$ CE, according to Table 2. LR uses a full random permutation rule instead of fixed-per-PGD-step like the rest, so it should be compared to L only and is, in fact, worse than L (Table 3). Stronger attacks are generally associated with higher cosine similarity, because if subsequent gradient directions agree more, then the optimization trajectory will be less erratic. Similarly, stronger attacks are also correlated with lower normalized variance and lower CV, indicating that there is less (normalized) variation among their gradient

| Source/Target Models | Normal | AT | RT |
|---|---|---|---|
| Normal | 0.00 | 79.50 | $84.52 \pm 0.45$ |
| AT | 80.20 | 44.30 | $77.00 \pm 0.46$ |
| RT | 78.10 | 78.30 | $29.94 \pm 0.40$ |

Table 8: The accuracy of three models, a normally trained model (Normal), AT, and RT, on adversarial examples generated on the other two models (transfer attacks) as well as itself (white-box attack). We use $\epsilon = 16/255$, and the adversarial examples for Normal and AT are generated via 1000-step PGD attack with two random restarts.

samples.

On the other hand, the un-normalized variance plot in Fig. 9 is not indicative of this relationship because different adversarial objectives have different scales and so do their gradients. However, among the same loss function (CE vs. CS vs. CL or LR vs. L), the correlation between stronger attacks and smaller variance is still present. Additionally, note that for a given attack objective, increasing $n$ leads to smaller normalized variance and CV as well as larger cosine similarity as we would expect.

### D.2. Transfer Attacks

Table 8 shows that RT models are very robust against black-box transfer attacks from normal and AT models. This is additional proof that our implementation of RT does not suffer from severe gradient obfuscation. Adversarial examples generated by attacking RT also transfer slightly better than adversarial examples generated with the other two attacks.

## E. Additional Experiments on the RT Model

### E.1. Effects of Decision Rules and the Number of Monte Carlo Samples

Fig. 10 and Fig. 11 compare three different decision rules that aggregate the $n$ outputs of the RT model to produce the final prediction $\hat{y}(x)$ given an input $x$. We choose the average softmax probability rule for all of our RT models because it provides a good trade-off between the clean accuracy and the robustness. Majority vote has poor clean accuracy, and the average logits have poor robustness.

### E.2. Additional Experiments on the Transformation Groups' Importance

Table 9 gives an alternative way to gauge the contribution of each transformation group. Table 5 in the main paper compares the performance when one group of transformations is omitted. Here, Table 9 compares the performance when only one group is used. According to this experiment,

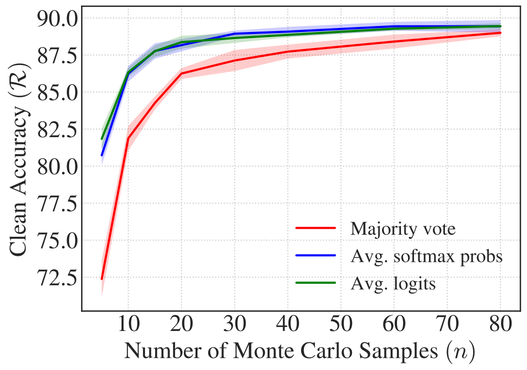


Figure 10: Clean accuracy of our best RT model computed with three decision rules for obtaining the final prediction from the $n$ output samples. The rules are majority vote (red), average softmax probability (blue), and average logits (green). The shaded areas represent the 95% confidence interval for each decision rule.

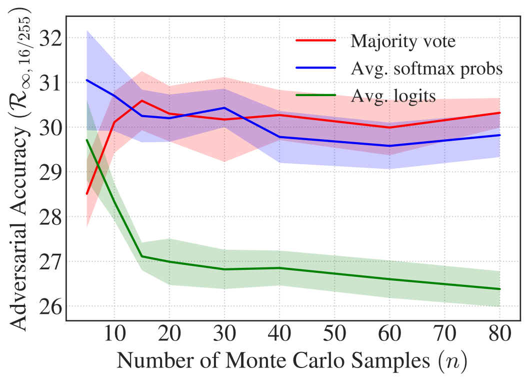


Figure 11: Adversarial accuracy ($\epsilon = 16/255$) of our best RT model computed with three decision rules for obtaining the final prediction from the $n$ output samples. The rules are majority vote (red), average softmax probability (blue), and average logits (green). The shaded areas represent the 95% confidence interval for each decision rule. We use the same set of adversarial examples for all the rules, and they are generated with our best attack (Linear+Adam+SGM) run for 1600 steps.

noise injection appears most robust followed by lossy compression and geometric transformations. However, this result is less informative compared to Table 5 since most of the groups have zero adversarial accuracy, and the rest are likely to also reduce to zero given more attack steps. This result also surprisingly follows the commonly observed robustness-accuracy trade-off [45].

We do not present the average values of fine-tuned $\alpha$ or $p$ because there is no straightforward way to relate their values to the strength of each transformation. For example, an $\alpha$

| Used Transformations | Clean Acc. | Adv. Acc. |
|---|---|---|
| Noise injection | $80.93 \pm 0.44$ | $\mathbf{8.35 \pm 0.20}$ |
| Blur filter | $97.32 \pm 0.20$ | $0.00 \pm 0.00$ |
| Color space | $94.40 \pm 0.53$ | $0.00 \pm 0.00$ |
| Edge detection | $97.64 \pm 0.09$ | $0.00 \pm 0.00$ |
| Lossy compression | $83.56 \pm 0.66$ | $3.56 \pm 0.26$ |
| Geometric transforms | $88.42 \pm 0.28$ | $0.83 \pm 0.21$ |
| Stylization | $\mathbf{98.31 \pm 0.09}$ | $0.00 \pm 0.00$ |

Table 9: RT's performance when only one of the transformation groups is applied. The attack is Linear+Adam+SGM with 200 steps and $n = 20$.

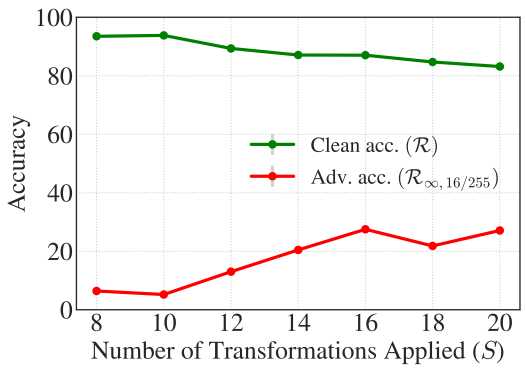


Figure 12: Adversarial accuracy of RT models obtained after running Algorithm 1 for different values of $S$.

of 0.5 for transform A might "change" the original image more than transform B with the same $\alpha$. As a general rule of thumb, we set the range of $\alpha$ such that $\alpha = 1$ is the most extreme parameter choice and completely destroys the image. For instance, $\alpha = 1$ for uniform noise would stretch the range to $[-1, 1]$, or $\alpha = 1$ for pepper noise would return an image with all zero pixels. On the other hand, setting $\alpha = 0$ means that output is always the original image.

## E.3. Effects of the Number of Transformations

We test the effect of the transform permutation size $S$ on the clean and the robust accuracy of RT models (Fig. 12). We run Bayesian optimization experiments for different values of $S$ using all 33 transformation types, and all of the models are trained using the same procedure. As mentioned in Section 7.2, a larger $S$ generally increases the robustness but decreases the clean accuracy. The improvement on the robustness plateaus after some value of $S$ (16 for Imagenette when sampling from all of our transformation types). We believe that the small bump at $S = 18$ is due to some variations in the performance of Bayesian optimization.

## E.4. Combination with Adversarial Training

In an attempt to further improve the robustness of our best RT model, we combine it with the adversarial training scheme. Specifically, we train an RT model with the best set of hyperparameters on adversarial examples generated with a 10-step PGD attack and $n = 4$. We use 10 PGD steps to follow the approaches taken by previous works as well as to be consistent with AT and TRADES models in this work. Due to computational constraints, it is difficult for us to use a stronger attack with more steps or a larger $n$. We also found that with the given number of steps and value of $n$, the adversarial objectives that we experimented with all perform roughly the same.

Table 10 compares our best RT model to its adversarially trained version (RT+AT). As mentioned in Section 7.2, we can see that RT+AT "overfits" to the specific attack it is trained on (PGD with linear loss) and achieves a slightly higher adversarial accuracy against that same attack. However, on the strongest attack, RT+AT is, in fact, less robust than the RT model alone by a considerable margin. Surprisingly, this result contradicts prior observations of normal neural networks whose robustness always improves with adversarial training even when the attack is extremely weak such as FGSM [46].

Our hypothesis is that the network will "memorize" or "overfit" to the few adversarial examples the weak attack is able to produce for the already defended RT model. This same phenomenon is also observed by Tramèr et al. [44] and Wong et al. [46] when an AT model overfits to FGSM attacks that are initialized incorrectly and produce a limited set of all possible adversarial examples. Given more time and compute, this hypothesis can be verified by adversarial training with a much stronger attack. Nonetheless, such a procedure would not be scalable unless we devise a much more efficient attack. We believe that this is an interesting future direction to further improve both the robustness of the attack and the RT model simultaneously.

| Attacks | Models | Adversarial Accuracy ($\mathcal{R}_{\infty,16/255}$) at Different PGD Steps | | | | |
|---|---|---|---|---|---|---|
| | | 100 | 200 | 500 | 800 | 1000 |
| Linear | RT | $71.63 \pm 0.35$ | $63.03 \pm 0.37$ | $58.19 \pm 0.57$ | $55.46 \pm 0.39$ | $54.08 \pm 0.44$ |
| | RT+AT | $71.11 \pm 0.62$ | $62.97 \pm 0.26$ | $59.37 \pm 0.46$ | $56.61 \pm 0.49$ | $55.28 \pm 0.49$ |
| Linear+Adam+SGM | RT | $66.95 \pm 0.47$ | $58.16 \pm 0.47$ | $44.17 \pm 0.56$ | $38.68 \pm 0.44$ | $36.91 \pm 0.54$ |
| | RT+AT | $66.97 \pm 0.61$ | $54.38 \pm 0.45$ | $39.47 \pm 0.44$ | $34.38 \pm 0.24$ | $33.21 \pm 0.68$ |

Table 10: Comparison of our best RT model and its combination with adversarial training (RT +AT) under two attacks. RT +AT model is trained with PGD attack with the linear loss ($n = 4$ and 10 steps).