Preprocessors Matter! Realistic Decision-Based Attacks on Machine Learning Systems

Chawin Sitawarin [1] **Florian Tramèr** [2] **Nicholas Carlini** [3]

Abstract

Decision-based attacks construct adversarial examples against a machine learning (ML) model by making only hard-label queries. These attacks have mainly been applied directly to standalone neural networks. However, in practice, ML models are just one component of a larger learning system. We find that by adding a *single* preprocessor in front of a classifier, state-of-the-art query-based attacks are up to *seven*$\times$ less effective at attacking a prediction pipeline than at attacking the model alone. We explain this discrepancy by the fact that most preprocessors introduce some notion of *invariance* to the input space. Hence, attacks that are unaware of this invariance inevitably waste a large number of queries to re-discover or overcome it. We therefore develop techniques to (i) reverse-engineer the preprocessor and then (ii) use this extracted information to attack the end-to-end system. Our preprocessors extraction method requires only a few hundreds queries, and our preprocessor-aware attacks recover the same efficacy as when attacking the model alone. The code can be found at `https://github.com/google-research/preprocessor-aware-black-box-attack`.

1. Introduction

Machine learning is widely used in security-critical systems, for example for detecting abusive, harmful or otherwise unsafe online content (Waseem et al., 2017; Clarifai; Jha & Mamidi, 2017). It is critical that such systems are robust against adversaries who seeks to evade them.

Yet, an extensive body of work has shown that an adversary can fool machine learning models with *adversarial examples* (Biggio et al., 2013; Szegedy et al., 2014). Most prior work focuses on *white-box* attacks, where an adversary has perfect knowledge of the entire machine learning system (Carlini & Wagner, 2017). Yet, real adversaries rarely have this level of access (Tramèr et al., 2019), and must thus instead resort to *black-box* attacks (Chen et al., 2017). *Decision-based* attacks (Brendel et al., 2018a) are a particularly practical attack vector, as these attacks only require the ability to query a target model and observe its decisions.

However, existing decision-based attacks (Brendel et al., 2018b; Cheng et al., 2020a; Chen et al., 2020; Li et al., 2020) have primarily been evaluated against standalone ML models "*in the lab*", thereby ignoring the components of broader learning systems that are used in practice. While some decision-based attacks have been demonstrated on production systems as a proof-of-concept (e.g., Ilyas et al. (2018); Brendel et al. (2018a); Li et al. (2020)), it is not well understood how these attacks perform on end-to-end learning systems compared to standalone models.

We show that **existing decision-based attacks are significantly less effective against end-to-end systems compared to standalone machine learning models.** For example, a standard decision-based attack can evade a ResNet image classifier on ImageNet with an average ℓ_2-distortion of 3.7 (defined formally later). Yet, if we instead attack an end-to-end learning system that simply *preprocesses* the classifier's input before classifying it—e.g., by resizing or compressing the image—the attack achieves an average ℓ_2 distortion of 28.5—**a $7\times$ increase!** We further find that extensive hyperparameter tuning and running the attacks for more iterations fail to resolve this issue. We thus argue that existing decision-box attacks have fundamental limitations that make them sub-optimal in practice.

To remedy this, **we develop improved attacks that achieve the same success rate when attacking systems with unknown preprocessors, as when attacking standalone models**. Our attacks combine decision-based attacks with techniques developed for model extraction (Tramèr et al., 2016). Our attacks first query the system to reverse-engineer the preprocessor(s) used in the input pipeline, and then mount a modified *preprocessor-aware* decision-based attack. Our extraction procedure is efficient and often requires only a few hundred queries to identify commonly used preproces-

[1]Department of Computer Science, University of California, Berkeley, USA. Work partially done while the author was at Google. [2]ETH Zürich, Zürich, Switzerland. [3]Google DeepMind, Mountain View, USA. Correspondence to: Chawin Sitawarin <chawins@berkeley.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

sors. This cost can also be amortized across many generated adversarial examples. We find that even the *least efficient* preprocessor-aware attack outperforms *all* unaware attacks. Learning the system's preprocessing pipeline is thus more important than devising an efficient standalone attack.

2. Background and Related Work

Adversarial Examples. Adversarial examples are inputs designed to fool a machine learning classifier (Biggio et al., 2013; Szegedy et al., 2014; Goodfellow et al., 2015). For some classifier f, an example x has an adversarial example $x' = x + \delta$ if $f(x) \neq f(x')$, where δ is a small perturbation under some ℓ_p-norm, i.e., $\|\delta\|_p \leq \epsilon$. Adversarial examples can be constructed either in the white-box setting (where the adversary uses gradient descent to produce the perturbation δ) (Carlini & Wagner, 2017; Madry et al., 2018), or more realistically, in the black-box setting (where the adversary uses just query access to the system) (Papernot et al., 2017; Chen et al., 2017; Brendel et al., 2018a). Our paper focuses on this black-box setting with ℓ_2-norm perturbations.

Decision-based can generate adversarial examples with only query access to the remote model's decisions (i.e., the output class $y \leftarrow f(x)$). These attacks typically work by finding the decision boundary between the original image and a target label of interest and then walking along the decision boundary to reduce the total distortion (Brendel et al., 2018a; Cheng et al., 2020a; Chen et al., 2020; Li et al., 2020).

It has been shown that decision-based attacks should operate at the lowest-dimensional input space possible. For example, QEBA (Li et al., 2020) improves upon HSJA (Chen et al., 2020) by constructing adversarial examples in a lower-dimensional embedding space. This phenomenon will help explain some of the results we observe, where we find that high-dimensional images require more queries to attack.

Adversarial examples need not exploit the classifier itself. *Image scaling attacks* (Quiring et al., 2020) construct a high-resolution image x so that after resizing to a smaller $\hat{x}$, the low resolution image is visually dissimilar to x. As a result, any accurate classifier will (correctly) classify the high-resolution image and the low-resolution image differently. Gao et al. (2022) consider the image-scaling attack in conjunction with a classifier similar to our setting. However, our work applies to arbitrary preprocessors, not limited to resizing, and we also propose an extraction attack to unveil the deployed preprocessor in the first place.

Preprocessing defenses. A number of proposed defenses against adversarial examples preprocess inputs before classification (Guo et al., 2018; Song et al., 2018). Unfortunately, these defenses are largely ineffective in a white-box setting (Athalye et al., 2018; Tramer et al., 2020; Sitawarin et al., 2022). Surprisingly, recent work has shown that defending against existing decision-based attacks with preprocessors is quite simple. Aithal & Li (2022); Qin et al. (2021) show that adding small amounts of random noise to inputs impedes all current attacks. This suggests that there may be a significant gap between the capabilities of white-box and black-box attacks when preprocessors are present.

Model Stealing Attacks. To improve the efficacy of black-box attacks, we make use of techniques from model stealing attacks (Tramèr et al., 2016). These attacks aim to create a ML model that closely mimics the behavior of a remote model (Jagielski et al., 2020). Our goal is slightly different as we only aim to "steal" the system's preprocessor and use this knowledge to mount stronger evasion attacks. For this, we leverage techniques that have been used to extract *functionally equivalent* models, which exactly match the behavior of the remote model on all inputs (Milli et al., 2019; Rolnick & Kording, 2020; Carlini et al., 2020).

3. Setup and Threat Model

3.1. Notation

We denote an unperturbed input image in the *original space* as $x_o \in \mathcal{X}_o \coloneqq [0,1]^{s_o \times s_o}$ and a processed image in the *model space* as $x_m \in \mathcal{X}_m \subseteq [0,1]^{s_m \times s_m}$. The original size s_o can be the same or different from the target size s_m. A preprocessor $t : \mathcal{X}_o \to \mathcal{X}_m$ maps x_o to $x_m \coloneqq t(x_o)$. For instance, a resizing preprocessor that maps an image of size 256×256 pixels to 224×224 pixels means that $s_o = 256$, $s_m = 224$, and $\mathcal{X}_m = [0,1]^{224 \times 224}$. As another example, an 8-bit quantization restricts $\mathcal{X}_m$ to a discrete space of $\{0, 1/255, 2/255, \ldots, 1\}^{s_m \times s_m}$ and $s_o = s_m$. The classifier, excluding the preprocessor, is represented by $f : \mathcal{X}_m \to \mathcal{Y}$ where $\mathcal{Y}$ is the hard label space. Finally, the entire classification pipeline is denoted by $f \circ t : \mathcal{X}_o \to \mathcal{Y}$.

3.2. Threat Model

The key distinguishing factor between previous works and ours is that we consider a **preprocessing pipeline as part of the victim system**. In other words, the adversary cannot simply run an attack algorithm on the model input space. We thus follow in the direction of Pierazzi et al. (2020) and Gao et al. (2022) who develop attacks that work end-to-end, as opposed to just attacking a standalone model. To do this, we develop strategies to "bypass" the preprocessors (Section 4) and to reverse-engineer which preprocessors are being used (Section 6). Our threat model is:

- The adversary has *black-box, query-based* access to the victim model and can query the model on any input and observe the output label $y \in \mathcal{Y}$. The adversary has a limited query budget per input. The adversary knows nothing else about the system.

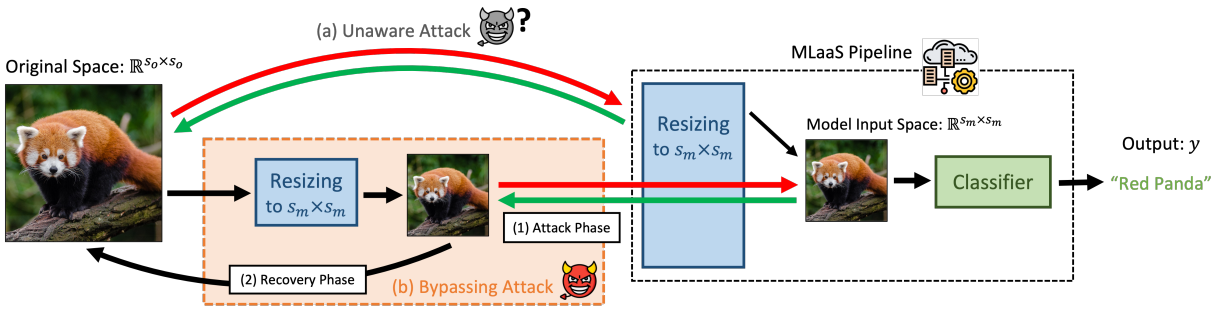


Figure 1: Illustration of our Bypassing Attack with resizing as the preprocessor as a comparison to the unaware or preprocessor-oblivious attack. The red and the green arrows denote the query submitted by the attack and the output returned by the MLaaS pipeline, respectively. The attack phase of our Bypassing Attack first resizes the input image to the correct size used by the target pipeline. This allows any attack algorithm to operate on the model input space directly. The recovery phase then finds the adversarial example in the original space that maps to the one found during the attack phase.

- The adversary wants to misclassify as many perturbed inputs as possible (either targeted and untargeted), while minimizing the perturbation size—measured by Euclidean distance in *the original input space* $\mathcal{X}_o$.
- The victim system accepts inputs of any dimension, and the desired model input size is obtained by cropping and/or resizing as part of an image preprocessing pipeline.

4. Preprocessor-Aware Attacks

Decision-based attacks often query a model on many nearby points, e.g., to approximate the local geometry of the boundary. Since most preprocessors are not *injective* functions, nearby points in the original input space might map onto the same processed image. Preprocessing thus makes the model's output *invariant* to some input changes. This can cause the attack to waste queries and prevent it from learning information about the target model.

4.1. Bypassing Attack

Our Bypassing Attack in Algorithm 1 avoids these invariances by circumventing the preprocessor entirely. Figure 1 illustrates our attack with a resizing preprocesssor (e.g., $1024 \to 224$). To allow the Bypassing Attack to query the model directly, we first map the input image ($x_o \in \mathcal{X}_o$) to the preprocessed space ($t(x_o) \in \mathcal{X}_m$). Then, in the *Attack Phase*, we execute an off-the-shelf decision-based attack directly on this preprocessed image ($x_m^{\text{adv}} \in \mathcal{X}_m$).

Finally, after completing the attack, we recover the adversarial image in the original space ($x_o^{\text{adv}} \in \mathcal{X}_o$) from x_m^{adv}. We call this step the *Recovery Phase*. It finds an adversarial example with minimal perturbation in the original space, by solving the following optimization problem:

$$\arg\min_{z_o \in \mathcal{X}_o} \|z_o - x_o\|_2^2 \quad \text{s.t.} \quad t(z_o) = x_m^{\text{adv}} \,. \tag{1}$$

Algorithm 1 Outline of Bypassing Attack. This example is built on top of a gradient-approximation-based attack algorithm (e.g., HSJA, QEBA), but it is compatible with any black-box attack. `ApproxGrad()` and `AttackUpdate()` are unmodified gradient approximation and perturbation update functions from the base attack. $\mathcal{U}$ is a distribution of vectors on a uniform unit sphere.

Input: Image x, label y, classifier f, preprocessor t
Output: Adversarial examples x^{adv}
$x' \leftarrow t(x)$ # *Initialization*
# ***Attack Phase**: run an attack algorithm of choice*
for $i = 1$ **to** `num_steps` **do**
  $\tilde{X} \leftarrow \{x' + \alpha u_b\}_{b=1}^{B}$ where $u_b \sim \mathcal{U}$
  $\nabla_x S \leftarrow \texttt{ApproxGrad}(f \circ t, \tilde{X}, y)$
  $x' \leftarrow \texttt{AttackUpdate}(x', \nabla_x S)$
end for
# ***Recovery Phase**: exactly recover x^{adv} in original input space*
$x^{\text{adv}} \leftarrow \texttt{ExactRecovery}(t, x')$
return x^{adv}

4.1.1. Cropping

Because almost all image classifiers operate on square images (Wightman, 2019), one of the most common preprocessing operations is to first crop the image to a square. In practice, this means that any pixels on the edge of the image are completely ignored by the classifier. Our Bypassing Attack exploits this fact by simply removing these cropped pixels, simply running an off-the-shelf attack in the cropped space. For a more formal statement, see Appendix B.1.

4.1.2. Resizing

Image resizing is a ubiquitous preprocessing step in any vision system, as most classifiers are trained only on images of a specific size. We begin by considering the special case

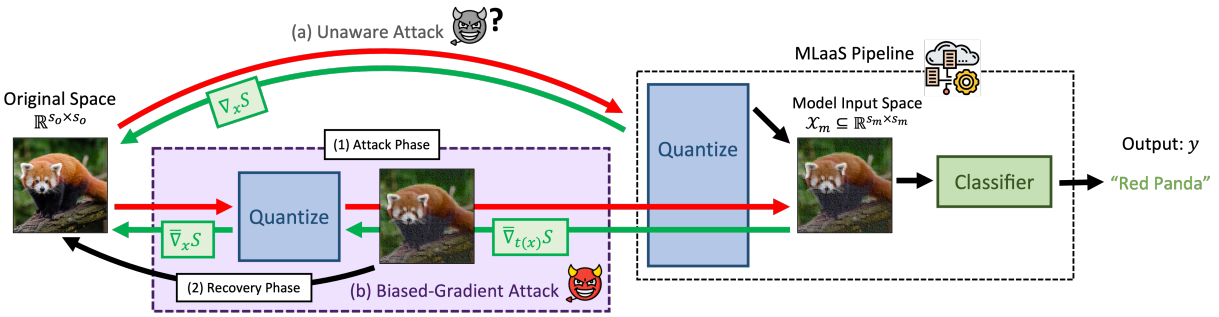


Figure 2: Illustration of the Biased-Gradient Attack with quantization as the preprocessor. Biased-Gradient Attack cannot directly operate on the model input space like Bypassing Attack. Rather, it takes advantage of the preprocessor knowledge by modifying a specific attack but still operates in the original space.

of resizing with "nearest-neighbor interpolation", which downsizes images by a factor k simply by selecting only 1 out of every block of k pixels. This resize operation is conceptually similar to cropping, and thus the intuition behind our attack is the same: if we know which pixels are retained by the preprocessor, we can avoid wasting perturbation budget and queries on pixels that are discarded. Other interpolation methods for resizing, e.g., bilinear or bicubic, work in a similar way, and can all be expressed as a linear transform, i.e., $x_m = t^{\text{res}}(x_o) = M^{\text{res}} x_o$ for $s_o > s_1$.

The attack phase for resizing is exactly the same as that of cropping. The adversary simply runs an attack algorithm of their choice on the model space $\mathcal{X}_m$. The main difference comes in the recovery phase which amounts to solving the following optimization problem:

$$x_o^{\text{adv}} = \arg\min_{z_o \in \mathbb{R}^{s_o \times s_o}} \|z_o - x_o\|_2 \text{ s.t. } M^{\text{res}} z_o = x_m^{\text{adv}}. \quad (2)$$

Quiring et al. (2020); Gao et al. (2022) solve a similar version of this problem via a gradient-based algorithm. However, we show that there exists a closed-form solution for the global optimum. Since the constraint in Eqn. (2) is an underdetermined linear system, this problem is analogous to finding a minimum-norm solution, given by:

$$x_o^{\text{adv}} = x_o + \delta_o^* = x_o + (M^{\text{res}})^+ \left(x_m^{\text{adv}} - x_m\right). \quad (3)$$

Here, $(\cdot)^+$ represents the Moore-Penrose pseudo-inverse. We defer the formal derivation to Appendix B.2.

Limitation. We have demonstrated how to bypass two very common preprocessors—cropping and resizing—but not all can be bypassed in this way. Our Bypassing Attack assumes **(A1)** the preprocessors are *idempotent*, i.e., $t(t(x)) = t(x)$, and **(A2)** the preprocessor's output space is continuous. Most common preprocessing functions are idempotent: e.g., quantizing an already quantized image makes no difference. For preprocessors that do not satisfy **(A2)**, e.g., quantization whose output space is discrete, we propose an alternative attack in the next section.

4.2. Biased-Gradient Attacks

We now turn our attention to more general preprocessors that cannot be bypassed without modifying the search space—for example quantization, discretizes a continuous space. Quantization is one of the most common preprocessors an adversary has to overcome since all common image formats (e.g., PNG or JPEG) discretize the pixel values to 8 bits. However, prior black-box attacks ignore this fact and operate in the continuous domain.

We thus propose the Biased-Gradient Attack in Algorithm 2. Unlike the Bypassing Attack, this attack operates in the original space. Instead of applying a black-box attack as is, the Biased-Gradient Attack modifies the base attack in order to bias queries toward directions that the preprocessor is more sensitive to. The intuition is that while it is hard to completely avoid the invariance of the preprocessor, we can encourage the attack to explore directions that result in large changes in the output space of the preprocessing function.

Our Biased-Gradient Attack also consists of an attack and recovery phase. The attack phase makes two modifications to an underlying gradient approximation attack (e.g., HSJA, QEBA) which we explain below. The recovery phase simply solves Equation (1) with a gradient-based method, by relaxing the constraint using a Lagrange multiplier (since closed-formed solutions do not exist in general). For this, we defer the details to Appendix C. Figure 2 illustrates the Biased-Gradient Attack for a quantization preprocessor.

(i) Biased Gradient Approximation: We modify the gradient approximation step to account for the preprocessor. First, consider the adversary's loss function defined as

$$S(x) := \begin{cases} \max_{c \in \mathcal{Y} \setminus \{y\}} f_c(x) - f_y(x) & \text{(untargeted)} \\ f_{y'}(x) - \max_{c \in \mathcal{Y} \setminus \{y'\}} f_c(x) & \text{(targeted)} \end{cases} \quad (4)$$

where (x, y) is the input, and $y' \neq y$ is the target label. Attacks such as HSJA and QEBA estimate the gradient of

Algorithm 2 Outline of Biased-Gradient Attack built on top of gradient-approximation-based attack algorithm.

Input: Image x, label y, classifier f, preprocessor t
Output: Adversarial examples x^{adv}
$x' \leftarrow x$ *# No special initialization*
*# **Attack Phase**: run modified attack*
for $i = 1$ **to** `num_steps` **do**
  # Biased gradient approximation
  $\tilde{X} \leftarrow \{t(x' + \alpha u_b)\}_{b=1}^{B}$ where $u_b \sim \mathcal{U}$
  $\nabla_{t(x)} S \leftarrow \texttt{ApproxGrad}(f \circ t, \tilde{X}, y)$
  $\bar{\nabla}_x S \leftarrow \bar{\nabla}_{t(x)} S \cdot \frac{\partial t(x)}{\partial x}$ *# Backprop through t*
  $x' \leftarrow \texttt{AttackUpdate}(x', \nabla_x S)$
end for
*# **Recovery Phase**: optimization-based recover x^{adv} in original space (works for any differentiable t)*
$x^{\text{adv}} \leftarrow \texttt{OptRecovery}(t, x')$
return x^{adv}

$S(x)$ by applying finite-differences to the quantity $\phi(x) \coloneqq \text{sign}(S(x))$ which can be measured by querying the model's label. The attack samples uniformly random unit vectors $\{u_b\}_{b=1}^{B}$, scales them a hyper-parameter α, and computes

$$\nabla_x S(x, \alpha) \approx \frac{1}{B} \sum_{b=1}^{B} \phi(t(x + \alpha u_b)) u_b \ , \tag{5}$$

We then perform a change-of-variables to obtain a gradient estimate with respect to $t(x)$ instead of x:

$$\frac{1}{B} \sum_{b=1}^{B} \phi(t(x + \alpha u_b)) u_b = \frac{1}{B} \sum_{b=1}^{B} \phi(t(x) + \alpha'_b u'_b) u_b \tag{6}$$

where $\alpha'_b = \|t(x + \alpha u_b) - t(x)\|_2$, and $u'_b = (t(x+\alpha u_b) - t(x))/\alpha'_b$. Notice that $\alpha'_b u'_b$ corresponds to a random perturbation in the model space. Thus, we can "bypass" the preprocessor and approximate gradients in the model space instead by substituting u_b with u'_b in Equation (6).

$$\bar{\nabla}_{t(x)} S(x, \alpha) \coloneqq \tfrac{1}{B} \textstyle\sum_{b=1}^{B} \phi(t(x) + \alpha'_b u'_b) u'_b \approx \nabla_{t(x)} S(x, \alpha). \tag{7}$$

So instead of querying the ML system with inputs $x + \alpha u_b$, we use $t(x+\alpha u_b) = t(x) + \alpha'_b u'_b$ which is equivalent to pre-applying the preprocessor to the queries. If the preprocessor is idempotent, the model f sees the same processed input in both cases. This gradient estimator is biased because u'_b depends on t. Concretely, the distribution of u'_b is concentrated around directions that "survive" the preprocessor.

(ii) Backpropagate Gradients through the Preprocessor: The gradient estimate $\bar{\nabla}_{t(x)} S$ in Eqn. (7) is w.r.t. the model space, instead of the original input space where the attack operates. Hence, we can backpropagate $\bar{\nabla}_{t(x)} S$ through t according to the chain rule, $\bar{\nabla}_x S = \nabla_x t(x) \cdot \bar{\nabla}_{t(x)} S$ where $\nabla_x t(x)$ is the Jacobian matrix of the preprocessor t w.r.t. the original space. In our experiments, we use a differentiable version of quantization and JPEG compression by Shin & Song (2017) so the Jacobian matrix exists.

5. Attack Experiments

5.1. Setup

Model. Similarly to previous works (Brendel et al., 2018a), we evaluate our attacks on a ResNet-18 (He et al., 2016) trained on the ImageNet dataset (Deng et al., 2009). The model is publicly available in the popular `timm` package (Wightman, 2019).

Off-the-shelf attacks. We consider four different attacks, Boundary Attack (Brendel et al., 2018a), Sign-OPT (Cheng et al., 2020a), HopSkipJump Attack (HJSA) (Chen et al., 2020), and QEBA (Li et al., 2020). The first three attacks have both targeted and untargeted versions while QEBA is only used as a targeted attack. We also compare our attacks to the baseline preprocessor-aware attack, SNS (Gao et al., 2022). As this attack only considers resizing, we adapt it to the other preprocessors we consider.

Attack hyperpameters. As we discuss in Section 7.2 and Appendix E.2, a change in preprocessor has a large impact on the optimal choice of hyperparameters for each attack. We thus sweep hyperparameters for all attacks and report results for the best choice.

Metrics. We report the average perturbation size (ℓ_2-norm) of adversarial examples found by each attack—referred to as the "adversarial distance" in short. Smaller adversarial distance means a stronger attack.

Appendix A contains full detail of all our experiments.

5.2. Bypassing Attack Results

Cropping. We consider a common operation that center crops an image of size 256×256 pixels down to 224×224 pixels, i.e., $s_o = 256, s_m = 224$. In Table 1, our Bypassing approach improves all of the baseline preprocessor-unaware attacks. The adversarial distance found by the baseline is about 8–16% higher than that of the Bypassing Attack counterpart across all settings. This difference is very close to the portion of the border pixels that are cropped out ($\sqrt{256^2/224^2} - 1 \approx 0.14$), suggesting that the cropping-unaware attacks do waste perturbation on these invariant pixels. Our Bypassing Attack also recovers about the same mean adversarial distance as the case where there is no preprocessor (first row of Table 1).

Resizing. We study the three most common interpolation or resampling techniques, i.e., nearest, bilinear, and bicubic. For an input size of 1024×1024 (see Table 1), a reason-

Table 1: Comparing the mean adversarial perturbation norm (↓) computed by preprocessor-unaware attacks vs our Bypassing Attack for the classifier without any preprocessor and with cropping and resizing preprocessors. Lower is better, and the best results for untargeted and targeted settings are in bold.

Preprocessors	Methods	Untargeted Attacks			Targeted Attacks			
		Boundary	Sign-OPT	HSJA	Boundary	Sign-OPT	HSJA	QEBA
None	n/a	4.6	5.7	**3.6**	36.7	45.6	32.2	**19.1**
Crop ($256 \to 224$)	Unaware	5.3	6.5	4.2	42.8	52.7	38.2	22.2
	Bypass (ours)	4.6	5.8	**3.6**	37.3	46.3	32.9	**19.6**
Resize (Nearest)	Unaware	21.2	24.8	16.5	172.2	198.8	153.4	90.5
	Bypass (ours)	4.7	5.8	**3.7**	37.7	46.3	33.3	**19.4**
Resize (Bilinear)	Unaware	32.7	38.2	25.5	198.3	213.0	188.4	90.3
	Bypass (ours)	7.4	9.1	**6.0**	58.2	70.9	50.3	**30.0**
Resize (Bicubic)	Unaware	25.7	29.2	20.6	184.8	207.3	171.6	91.2
	Bypass (ours)	5.8	7.1	**4.5**	46.4	57.7	40.6	**23.8**

able image size captured by digital or phone cameras, **our attack reduces the mean adversarial distance by up to $4.6\times$ compared to the preprocessor-oblivious counterpart.** For all image sizes we experiment with, including 256 and 512 pixels in Table 7, Bypassing Attack is always preferable to the resizing-oblivious attack both with and without hyperparameter tuning.

The improvement from the Bypassing Attack is proportional to the original input dimension. The benefit diminishes with a smaller original size because the base attack of the Bypassing Attack operates in the model space. Hence, it minimizes the adversarial distance in that space, i.e., the distance between x_m^{adv} and $x_m = t(x_o)$. This distance is likely correlated but not necessarily the same as the true objective distance measured in the original space, i.e., the distance between x_o^{adv} and x_o. In these cases, it may be preferable to use the Biased-Gradient Attack instead. The results are shown in Table 2 and Section 5.3.

5.3. Biased-Gradient Attack Results

We evaluate the Biased-Gradient Attack on a broad range of preprocessors; in addition to resize and crop, we include 8/6/4-bit quantization as well as JPEG compression with quality values of 60, 80, and 100. Moreover, we experiment with neural-network-based compression methods from Ballé et al. (2018) and Cheng et al. (2020b) as well as SwinIR, a transformer-based denoiser (Liang et al., 2021). We select these methods as representatives of the recent image restoration/compression models which improve upon the traditional computer vision techniques (Zhang et al., 2021; Zamir et al., 2022). Importantly, these methods violate our idempotent assumption so they serve an extra purpose of evaluating our attack when the assumption does not hold.

Here, we consider untargeted/targeted HSJA and targeted QEBA as they are consistently the strongest, and the other two do not involve gradient approximation. From Table 2, Biased-Gradient Attack outperforms the preprocessor-unaware counterpart as well as SNS in almost all settings. A few highlights are: **Biased-Gradient Attack reduces the mean adversarial distance to only $\frac{1}{3}$ and $\frac{1}{6}$ of the distance found by the attack without it for 4-bit quantization and JPEG with a quality of 60, respectively.** The Biased-Gradient Attack also outperforms the baselines on neural compression under varying models and compression levels as well as on the SwinIR denoiser (Table 3). We observe a recurring trend where the benefit of Biased-Gradient Attack increases with *stronger* preprocessors, e.g., fewer

Table 2: Comparison of the mean adversarial perturbation norm (↓) found by our Biased-Gradient Attacks vs the preprocessor-unaware and the SNS counterparts.

Preprocess	Methods	Untg.	Targeted	
		HSJA	HSJA	QEBA
Crop ($256 \to 224$)	Unaware	4.2	38.2	22.2
	SNS	3.7	35.4	31.5
	Biased-Grad (ours)	**3.7**	33.1	**19.6**
Resize ($1024 \to 224$) (Nearest)	Unaware	16.5	153.4	90.5
	SNS	3.9	112.6	32.2
	Biased-Grad (ours)	**3.7**	23.5	**19.4**
Quantize (4 bits)	Unaware	9.7	63.7	56.4
	SNS	6.4	55.9	57.2
	Biased-Grad (ours)	**3.1**	39.3	**28.8**
JPEG (quality 60)	Unaware	9.2	63.2	52.7
	SNS	2.7	44.5	44.6
	Biased-Grad (ours)	**1.5**	25.1	**21.0**
Neural Compress (Ballé et al., 2018) (hyperprior, 8)	Unaware	25.1	92.0	78.6
	SNS	17.6	83.6	78.9
	Biased-Grad (ours)	**15.8**	**75.2**	75.8
Neural Compress (Cheng et al., 2020b) (attention, 6)	Unaware	33.8	94.1	86.9
	SNS	14.3	80.3	75.5
	Biased-Grad (ours)	**12.6**	**74.8**	77.9

Table 3: The mean adversarial distance (↓) found by untargeted HSJA on various neural-network-based preprocessors. In the parentheses are the model type and compression level.

Neural Preprocessors	Unaware	SNS	Biased-Grad
Ballé et al. (2018) (hyperprior, 8)	25.1	17.6	**15.8**
Ballé et al. (2018) (hyperprior, 6)	28.7	17.0	**14.0**
Ballé et al. (2018) (factorized, 8)	24.0	15.1	**13.9**
Ballé et al. (2018) (factorized, 6)	26.9	10.4	**11.7**
Cheng et al. (2020b) (attention, 6)	25.7	12.6	**14.3**
Cheng et al. (2020b) (attention, 4)	31.3	13.7	**13.4**
Cheng et al. (2020b) (anchor, 6)	27.3	8.7	**7.0**
Cheng et al. (2020b) (anchor, 4)	32.7	7.8	**6.6**
SwinIR (denoise level 15)	24.4	55.0	**10.5**

quantization bits or lower compression quality.

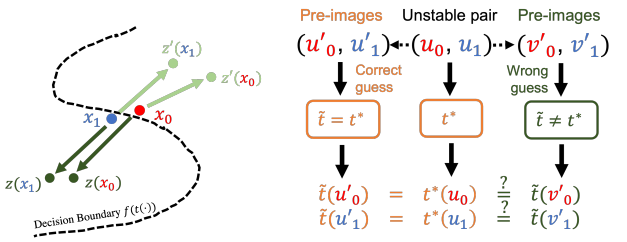


Figure 3: (Left) An *unstable example pair*, u_0, u_1. When slightly perturbed (either $z(\cdot)$ or $z'(\cdot)$), at least one of them is very likely to land on a different side of the decision boundary. (Right) Correct guess on the preprocessor t^* will yield pre-images, u'_0, u'_1, that map to the same image as the unstable pair. A wrong guess will not and likely result in a change in at least one of the predictions.

6. Extracting Preprocessors

As we have seen, knowledge of the preprocessor results in much more efficient decision-based attacks. What is now left is to design a query-efficient attack that actually reverse-engineers the preprocessor used by the target system.

It should not be surprising that this task would be achievable as it is a particular instance of the more general problem of *model stealing*. Recent work (Milli et al., 2019; Rolnick & Kording, 2020; Carlini et al., 2020) has shown a way to completely recover a (functionally-equivalent) neural network using only query access; stealing just a specific part of the model should thus be easier. Nonetheless, our setting comes with different challenges, both of which relate to the assumed adversary's capabilities:

1. Prior extraction attacks require *high-precision access* to the classifier, i.e., (64-bit) floating-point input/output. However, we can only provide valid image files (8-bit) as input and receive only a single decision label as output. This invalidates the approaches used in prior work that rely on computing finite differences with epsilon-sized input-output perturbations (Milli et al., 2019).
2. Prior attacks needs 10^3–10^7 queries to extract a very simple (10^3 parameters) MNIST neural network (Carlini et al., 2020)—in contrast we work with much larger models. While the up-front extraction cost can be amortized across many generated adversarial examples, for our attacks to be economically efficient, they must be effective in just a few hundred queries.

Intuition. Our extraction attack relies on a *guess-and-check strategy*. Given a hypothesis about the preprocessor (e.g., "the model uses bilinear resizing to 224×224 pixels"), we build a set of inputs Q such that the outputs $\{F(q) \mid q \in Q\}$ let us distinguish whether the hypothesis is true or not. Then, by enumerating a set $\mathbb{T}$ of possible preprocessors, we can use a combination of binary and exhaustive search to reduce this set down to a single preprocessor $t \in \mathbb{T}$. Our attack (see Algorithm 3) consists of two main components, "unstable pairs" and "pre-images," which we describe below.

6.1. Unstable Example Pairs

The first step of our attack generates an "unstable pair". This is a pair of samples u_0, u_1 with two properties: (i) $f(t(u_0)) \neq f(t(u_1))$, and (ii) with high probability p, $f(t(z(u_0))) = f(t(z(u_1)))$ for some random perturbation $z : \mathcal{X}_o \to \mathcal{X}_o$. Section 6.1 depicts an unstable pair: the points $x_0 \coloneqq t(u_0)$ and $x_1 \coloneqq t(u_1)$ have opposite labels, but a small random perturbation z is likely to push both points to the same side of the boundary. As the perturbation made by z (i.e., $z(u) - u$) grows, p should also increase.

Given two images x_0, x_1 such that $f(t(x_0)) \neq f(t(x_1))$, we construct an unstable pair (u_0, u_1) by performing two binary searches. The first finds a new pair of images (x'_0, x'_1) with $\|x'_0 - x'_1\|_0 = 1$. Starting from (x'_0, x'_1), the second binary search finds the unstable pair (u_0, u_1) where $\|u_0 - u_1\|_\infty = 1$. This gives us a pair of images that differ by one pixel, and by $1/255$ at that pixel, while also being predicted as different classes. This process uses only about 40 queries, depending on the input size. In the interest of space, we provide the detail in Appendix D.1. The rest of the attack uses only the unstable pair; x_i are no longer used.

6.2. Hypothesis Testing with Pre-Images

Suppose we hypothesize that the preprocessor applied to an image is some function $\tilde{t}$ (this is our "guess" piece of our guess-and-check attack). Then, given this unstable example pair (u_0, u_1), we can now implement the "check" piece. For clarity, we denote the actually deployed preprocessor by t^*.

We begin by constructing a *pre-image* $u'_0 \neq u_0$ so that $\tilde{t}(u_0) = \tilde{t}(u'_0)$ and analogously for u_1 and u'_1. Now if our guess is indeed correct, then it is guaranteed that $\forall i \in$

Table 4: Number of queries (mean ± standard deviation) necessary to determine what preprocessor is being used.

Preprocessor Space	Num. Queries
Arbitrary resize (200px–800px)	632 ± 543
Arbitrary center crop (0%-100%)	52.0 ± 1.3
Arbitrary JPEG compression (quality 50-100)	70.0 ± 22.8
Typical resize (see text)	48.7 ± 6.8

$\{0,1\}$ $F(u_i') = F(u_i)$ since $\tilde{t}(u_0') = \tilde{t}(u_0) = t^*(u_0)$. On the other hand, if our guess is wrong, then we have $\exists i \in \{0,1\}$ $F(u_i') \neq F(u_i)$ with at least some probability p.

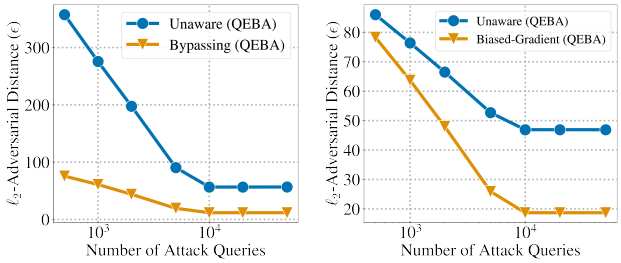
Let the null hypothesis be $\tilde{t} \neq t^*$. When we observe that the predictions of the pre-images do not change, we reject the null hypothesis if $1 - p \leq \alpha$ for some threshold α (we choose 0.01). To increase p, we can do two things: (i) simply repeat multiple trials by randomly generating and testing more pre-images and only reject the null hypothesis when *none* of the predictions change, or (ii) increase the size of the perturbation $u_i' - u_i$. This increases p by definition of the unstable pair.

6.3. Experiment on Real-World Applications

We use this attack to extract preprocessors for a wide range of models publicly hosted on HuggingFace Hub through their API (`https://huggingface.co/docs/api-inference`). We choose HuggingFace as the preprocessing metadata is available on most models for us to verify our extracted results. For each experiment, we randomly sample 10 models trained to predict ImageNet-1k classes. Table 4 summarizes the results.

Because our procedure is inherently guess-and-check, we first define the space of all possible preprocessors. The exact space here depends on the possible knowledge an adversary might have. In the worst case, an adversary has to enumerate over all possible image sizes ranging from the smallest size used for any image classifier (200px) to the largest size used for any image classifier (800px). This incurs a cost of 632 queries on average to extract one resizing operator (i.e., both output size and interpolation method). However, some preprocessors might be more *typical* than others. We call a preprocessor pipeline "typical" if it is used by at least two different models. For example, ResNet classifiers almost always first resize images to 256 × 256, and then center-crop the resulting image down to 224 × 224. Our set of typical sizes includes 224, 248, 256, 288, 299, 384, and 512 pixels, with bilinear and bicubic interpolations. With this prior knowledge, the adversary reduces the query cost by over 10 times, down to only ∼50 queries

Resize extraction is particularly efficient (∼50 queries) as we can use a binary search to find the crop size, instead of an exhaustive search. Any wrong guessed crop size larger than the actual size will not change the prediction of the pre-images. This allows us to run a binary search where the guessed crop size shrinks when the predictions do not change and grows otherwise.



(a) Resize (1024→224, nearest) (b) JPEG (quality 60)

Figure 4: Mean adversarial distance vs the number of queries used by targeted QEBA on two preprocessors.

7. Discussion

7.1. Varying Number of Attack Iterations

Figure 4 plots the mean adversarial distance as a function of the number of queries for QEBA. Notice that the adversarial distance plateaus after around 10,000 queries, and the distance found by preprocessor-unaware attacks never reaches that of our preprocessor-aware attacks. This suggests that our attacks do not only improve the efficiency of the algorithms but also allow them to find closer adversarial examples that would have been completely missed otherwise. See Appendix E.3 for more details.

7.2. Choice of Attack Hyperparameters

Hyperparameter choice is important to the effectiveness of the attacks. **In many cases, using the right hyperparameters benefits more than using stronger attack algorithms.** Choosing the right hyperparameter usually improves the distance found by ∼1.5× and up to 15× in one case, depending on the attack algorithm. The knowledge of the preprocessor deployed helps in quickly narrowing down the range of good hyperparameters. In practice, an adversary would benefit from spending some queries to learn the preprocessors as well as to tune the hyperparameters if one plans to generate many adversarial examples. For detailed numerical results and discussion, see Appendix E.2.

7.3. Varying the Target Model

We have used the public pre-trained ResNet-18 model as the target model in our experiments, but the conclusion also holds for other models. We pick two models, EfficientNetV2-Tiny (Tan & Le, 2021) and DEIT3-

Table 5: Mean adversarial distance (↓) found by targeted QEBA when multiple preprocessors are chained together.

List of Preprocessors	Unaware	SNS	BG
Resize(1024→256),Crop(224),Quant(8)	122.4	144.9	**97.9**
Resize(1024→256),Crop(224),JPEG(60)	283.1	237.5	**153.7**
Resize(512→224),Quant(6)	89.8	90.9	**49.0**

Small (Touvron et al., 2022), with different architectures from ResNet-18, and run the attacks with the resizing (nearest, $1024 \rightarrow 288$) and JPEG (quality 60) preprocessors, respectively. The mean adversarial distance for EfficientNetV2-Tiny/DEIT3-Small are 134.2/95.9, 117.7/76.6, **32.3/48.1** with unaware, SNS, and our Biased-Gradient attacks (+ targeted QEBA), respectively. This corresponds to about $4\times$ and $2\times$ improvement over the unaware attack on the two models.

7.4. Multiple Preprocessors

Preprocessor-aware attacks. In practice, multiple preprocessors are used sequentially. In the case that all the preprocessors can be bypassed, e.g., resizing and cropping, we can bypass the entire pipeline by querying with an appropriate size and padding. The recovery phase can then be done in the reverse order that the preprocessors are applied. When at least one preprocessor is not bypassable, we can treat the entire pipeline as one preprocessor and apply the Biased-Gradient Attack. Table 5 shows attack results for three common combinations of preprocessors.

Extracting multiple preprocessors. With the above attack, it becomes trivial to extract multiple preprocessors by extracting each in turn. Suppose there are two preprocessors t_1 and t_2, we can first extract t_1 by subsuming t_2 as part of f, i.e., $f' \circ t_1 \coloneqq f \circ t_2 \circ t_1$, and then we move on to guess t_2 using the now revealed t_1 to construct the pre-images. In practice, it is actually even easier: the most common two transformations, resizing and cropping, are almost commutative (i.e., $\texttt{Crop}(\texttt{Resize}(x)) \approx \texttt{Resize}(\texttt{Crop}(x))$ albeit with different crop and resize parameters). This means that one could either extract cropping or resizing first and still end up with an equivalent overall preprocessor pipeline.

8. Conclusion

We have shown that decision-based attacks are sensitive to changes in preprocessors, to a surprising degree. To develop a strong attack in practice, **it is more important to get the preprocessor right than to use a stronger attack!** We propose an extraction attack for commonly used preprocessors and two decision-based attacks aimed to circumvent any preprocessor. Our approaches are more efficient than the prior work and yield a stronger attack. We believe that it is important for future work to carefully consider other implicit assumptions in the current adversarial ML literature that may not be true in practice. We hope that our analysis will inspire future work to further explore this direction.

Acknowledgement

The authors would like to thank David Wagner for helping with the presentation of the paper, Matthew Jagielski for wonderful discussion on the problem, and Alex Kurakin for comments on early draft of this paper.

A majority of this research was conducted when Chawin was at Google as a student researcher. For the remaining time at UC Berkeley, Chawin was supported by the Hewlett Foundation through the Center for Long-Term Cybersecurity (CLTC), by the Berkeley Deep Drive project, and by generous gifts from Open Philanthropy.

References

Aithal, M. B. and Li, X. Mitigating black-box adversarial attacks via output noise perturbation. *IEEE Access*, 10: 12395–12411, 2022.

Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 274–283, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR.

Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N. Variational image compression with a scale hyperprior. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Bégaint, J., Racapé, F., Feltman, S., and Pushparaja, A. CompressAI: A PyTorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In Blockeel, H., Kersting, K., Nijssen, S., and Železný, F. (eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 387–402, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40994-3.

Brendel, W., Rauber, J., and Bethge, M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018a.

Brendel, W., Rauber, J., Kurakin, A., Papernot, N., Veliqi, B., Salathé, M., Mohanty, S. P., and Bethge, M. Adversarial vision challenge. Technical report, 2018b.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017. doi: 10.1109/SP.2017.49.

Carlini, N., Jagielski, M., and Mironov, I. Cryptanalytic extraction of neural network models. In *Annual International Cryptology Conference*, pp. 189–218. Springer, 2020.

Chen, J., Jordan, M. I., and Wainwright, M. J. HopSkipJumpAttack: A query-efficient decision-based attack. *arXiv:1904.02144 [cs, math, stat]*, April 2020.

Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pp. 15–26, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 978-1-4503-5202-4. doi: 10.1145/3128572.3140448.

Cheng, M., Singh, S., Chen, P. H., Chen, P.-Y., Liu, S., and Hsieh, C.-J. Sign-OPT: A query-efficient hard-label adversarial attack. In *International Conference on Learning Representations*, 2020a.

Cheng, Z., Sun, H., Takeuchi, M., and Katto, J. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020b.

Clarifai. Best NSFW model for content detection using AI — clarifai. https://www.clarifai.com/models/nsfw-model-for-content-detection.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Gao, Y., Shumailov, I., and Fawaz, K. Rethinking image-scaling attacks: The interplay between vulnerabilities in machine learning systems. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7102–7121. PMLR, July 2022.

Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Guo, C., Rana, M., Cisse, M., and van der Maaten, L. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. Black-box adversarial attacks with limited queries and information. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2137–2146. PMLR, July 2018.

Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., and Papernot, N. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1345–1362. USENIX Association, August 2020. ISBN 978-1-939133-17-5.

Jha, A. and Mamidi, R. When does a compliment become sexist? analysis and classification of ambivalent sexism using twitter data. In *Proceedings of the Second Workshop on NLP and Computational Social Science*, pp. 7–16, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2902. URL `https://aclanthology.org/W17-2902`.

Li, H., Xu, X., Zhang, X., Yang, S., and Li, B. QEBA: Query-efficient boundary-based blackbox attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., and Timofte, R. SwinIR: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 1833–1844, October 2021.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Milli, S., Schmidt, L., Dragan, A. D., and Hardt, M. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pp. 1–9, 2019.

MMEditing Contributors. MMEditing: OpenMMLab image and video editing toolbox, 2022.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pp. 506–519, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 978-1-4503-4944-4. doi: 10.1145/3052973.3053009.

Pierazzi, F., Pendlebury, F., Cortellazzi, J., and Cavallaro, L. Intriguing properties of adversarial ML attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1332–1349, May 2020. doi: 10.1109/SP40000.2020.00073.

Qin, Z., Fan, Y., Zha, H., and Wu, B. Random noise defense against query-based black-box attacks. *Advances in Neural Information Processing Systems*, 34:7650–7663, 2021.

Quiring, E., Klein, D., Arp, D., Johns, M., and Rieck, K. Adversarial preprocessing: Understanding and preventing image-scaling attacks in machine learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1363–1380. USENIX Association, August 2020. ISBN 978-1-939133-17-5.

Rauber, J., Brendel, W., and Bethge, M. Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

Rolnick, D. and Kording, K. Reverse-engineering deep relu networks. In *International Conference on Machine Learning*, pp. 8178–8187. PMLR, 2020.

Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2019.

Shin, R. and Song, D. JPEG-resistant adversarial images. In *Machine Learning and Computer Security Workshop (Co-Located with NeurIPS 2017)*, Long Beach, CA, USA, 2017.

Sitawarin, C., Golan-Strieb, Z., and Wagner, D. Demystifying the adversarial robustness of random transformation defenses. In *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond*, 2022.

Song, Y., Kim, T., Nowozin, S., Ermon, S., and Kushman, N. PixelDefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv:1710.10766 [cs]*, May 2018.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Tan, M. and Le, Q. EfficientNetV2: Smaller models and faster training. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10096–10106. PMLR, July 2021.

Touvron, H., Cord, M., and Jégou, H. DeiT III: Revenge of the ViT, April 2022.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. Stealing machine learning models via prediction apis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pp. 601–618, USA, 2016. USENIX Association. ISBN 978-1-931971-32-4.

Tramèr, F., Dupré, P., Rusak, G., Pellegrino, G., and Boneh, D. AdVersarial: Perceptual ad blocking meets adversarial machine learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2005–2021, November 2019. doi: 10.1145/3319535.3354222.

Tramer, F., Carlini, N., Brendel, W., and Madry, A. On adaptive attacks to adversarial example defenses. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1633–1645. Curran Associates, Inc., 2020.

Waseem, Z., Davidson, T., Warmsley, D., and Weber, I. Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online*, pp. 78–84, Vancouver, BC, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-3012. URL `https://aclanthology.org/W17-3012`.

Wightman, R. PyTorch image models. GitHub, 2019.

Zamir, S. W., Arora, A., Khan, S., Hayat, M., Khan, F. S., and Yang, M.-H. Restormer: Efficient transformer for high-resolution image restoration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5728–5739, June 2022.

Zhang, K., Li, Y., Zuo, W., Zhang, L., Van Gool, L., and Timofte, R. Plug-and-play image restoration with deep denoiser prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6360–6376, 2021.

A. Detailed Experiment Setup

We use a pre-trained ResNet-18 model from a well-known repository `timm` (Wightman, 2019) which is implemented in PyTorch and trained on inputs of size 224×224. This model is fixed throughout all the experiments. The experiments are run on multiple remote servers with either Nvidia Tesla A100 40GB or Nvidia V100 GPUs.

Implementations of Boundary Attack and HSJA are taken from the `Foolbox` package (Rauber et al., 2017).[1] For Sign-OPT Attack and QEBA, we use the official, publicly available implementation.[2]

To compare effectiveness of the attacks, we report the average perturbation size (ℓ_2-norm) of the adversarial examples computed on 1,000 random test samples. We will refer to this quantity as the adversarial distance in short. Smaller adversarial distance means a stronger attack. Unless stated otherwise, all the attacks use 5,000 queries per one test sample.

The implementation of the neural compression models along with their weights is taken from Bégaint et al. (2020). These models only accept input sizes that are powers of two so we change the input size from the default 224×224 to 256×256 here. For SwinIR, we use the publicly available code and weights from MMEditing Contributors (2022). Backpropagating the gradients through a SwinIR model takes up a large amount of memory which we cannot fit into our GPUs. So we have to reduce the input size to 128×128 pixels. Note that due to this mismatch in the original input sizes, it is not recommended to compare the mean adversarial distances across these preprocessors.

A.1. Hyperparameter Sweep

We find that the choice of hyperparameters of the four attack algorithms plays an important role in their effectiveness, and it is not clear how an attacker would know apriori how to choose such hyperparameters. In reality, the adversary would benefit from spending some queries to tune the hyperparameters on a few samples. Coming up with the most efficient tuning algorithm is outside of the scope of this work. Nonetheless, we account for this effect by repeating all experiments with multiple choices of hyperparameters and reporting the results with both the best sets throughout the paper. We include some of the results with both the best and the default hyperparameters for comparisons in Table 6, Table 9, and Table 8.

For Boundary attack, we sweep the two choices of step size, one along the direction towards the original input and the other in the orthogonal direction. The default values are $(0.01, 0.01)$, respectively, and the swept values are $(0.1, 0.01)$, $(0.001, 0.01)$, $(0.01, 0.1)$, and $(0.01, 0.001)$.

For Sign-OPT attack, we consider the update step size α and the gradient estimate step size β. Their default values are $(0.2, 0.001)$ respectively, and we sweep the following values: $(0.2, 0.01)$, $(0.2, 0.0001)$, $(0.02, 0.001)$, and $(2, 0.01)$.

We only tune one hyperparameter for HSJA and QEBA attacks but with the same number of settings (five) as the other two attacks above. For HSJA, we tune the update step size γ by trying values of 10^1 (default), 10^2, 10^3, 10^4, and 10^5. The optimal value of γ is always at a higher range than 10^1, not smaller. Lastly, we search the ratio r that controls the latent dimension that QEBA samples its random noise from for gradient approximation. We search over $r = 2, 4, 8, 16, 32$.

The observed trends and the recommended hyperparameters are discussed further below in Appendix E.2.

B. Bypassing Attacks

Here, we provide additional details on the Bypassing Attack for cropping and resizing preprocessors.

B.1. Cropping Preprocessor

Attack Phase for Cropping. To bypass the cropping transformation, the attacker simply submits an already cropped input and runs any query-based attack algorithm in the space $\mathbb{R}^{s_m \times s_m}$ instead of $\mathbb{R}^{s_o \times s_o}$. Without any modification to the attack algorithm, it is able to operate directly on the model space as if there is no preprocessing.

Recovery Phase for Cropping. In order for the adversarial example obtained from the attack phase to be useful in input space, the adversary still has to produce an adversarial example in the original space with the smallest possible Euclidean

[1] We use code from the commit: `https://github.com/bethgelab/foolbox/commit/de48acaaf46c9d5d4ea85360cadb5ab522de53bc`.

[2] Sign-OPT attack: `https://github.com/cmhcbb/attackbox`. QEBA: `https://github.com/AI-secure/QEBA`.

distance to the original input. It should be obvious that for cropping, this operation simply equates to padding this adversarial example with the original edge pixels.

Formal Definition of Cropping's Recovery Phase. We now formally describe what it means to crop an image. Given an input image of size $s_o \times s_o$, a crop operation removes the edge pixels of any image larger than a specified size, denoted by $s_m \times s_m$, such that the output has the size $s_o \times s_o$. Given an (flattened) input image $x_o \in \mathbb{R}^{s_o \times s_o}$ and the cropped image $x_m \in \mathbb{R}^{s_m \times s_m}$, we can write cropping as the following linear transformation, when $s_o > s_m$,

$$x_m = M^{\text{crop}} x_o \tag{8}$$

where $M^{\text{crop}} \in \mathbb{R}^{s_o^2 \times s_m^2}$ is a sparse binary matrix. Each row of M^{crop} has exactly one entry being 1 at a position of the corresponding non-edge pixel while the rest are 0. Note that we drop the "color-channel" dimension for simplicity since most of the preprocessors in this paper is applied channel-wise. We are only interested in the scenario when $s_o > s_m$ because otherwise, the preprocessing simply becomes an identity function.

Let the adversarial example in the model space as obtained from the attack phase be $x_m^{\text{adv}} \in \mathbb{R}^{s_m \times s_m}$. The adversary can recover the corresponding adversarial example in the original space, $x_o^{\text{adv}} \in \mathbb{R}^{s_o \times s_o}$, by padding x_m^{adv} with the edge pixels of x_o.

It is simple to show that x_o^{adv} is a projection of x_o onto the set $\mathcal{T}^{\text{crop}}(x_m^{\text{adv}}) \coloneqq \{x \in \mathcal{X}_o \mid t^{\text{crop}}(x) = x_m^{\text{adv}}\}$, i.e.,

$$x_o^{\text{adv}} = \underset{x \in \mathcal{T}^{\text{crop}}(x_m^{\text{adv}})}{\arg\min} \|x - x_o\|_2^2 \tag{9}$$

Proof. We can split $\|x - x_o\|_2^2$ into two terms

$$\sum_{i \in E} ([x]_i - [x_o]_i)^2 + \sum_{i \notin E} ([x]_i - [x_o]_i)^2 \tag{10}$$

where E is a set of edge pixel indices. The second term is fixed to $\left\|x_m^{\text{adv}} - t^{\text{crop}}(x_o)\right\|_2^2$ for any $x \in \mathcal{T}^{\text{crop}}(x_m^{\text{adv}})$. When $x = x_o^{\text{adv}}$, the first term is zero because x_o^{adv} is obtained by padding x_m^{adv} with the edge pixels of x_o. Since the first term is non-negative, we know that x_o^{adv} is a unique global minimum of Eqn. (9). □

B.2. Resizing Preprocessor

B.2.1. Computing the Transformation Matrix M^{res}

Not all image resizing operations are the same; the main step that varies between them is called the "interpolation" mode. Interpolation determines how the new pixels in the resized image depend on (multiple) pixels in the original image. Generally, resizing represents some form of a weighted average. How the weights are computed and how many of the original pixels should be used varies by specific interpolation methods.

For nearest interpolation (zeroth order), M^{res} is a sparse binary matrix with exactly one 1 per row. For higher-order interpolations, a pixel in x_m can be regarded as a weighted average of certain pixels in x_o. Here, M^{res} is no longer binary, and each of its rows represents these weights which are between 0 and 1. For instance, since one pixel in a bilinear resized image is a weighted average of four pixels (2×2 pixels) in the original image, M^{res} for bilinear interpolation has four non-zero elements per row. On the other hand, M^{res} for bicubic interpolation has 16 non-zero elements per row (4×4 pixels). M^{res} is still generally sparse for $s_o > s_1$ and is more sparse when s_o/s_1 increases.

The matrix M^{res} can be computed analytically for any given s_o and s_1. Alternatively, it can be populated programmatically, by setting each pixel in the original image to 1, one at a time, then performing the resize, and gathering the output. This method is computationally more expensive but simple, applicable to any sampling order, and robust to minor differences in different resizing implementations.

B.2.2. Recovery Phase for Resizing

The recovery phase involves some amount of linear algebra, as it is equivalent to solving the following linear system of equations

$$x_m^{\text{adv}} = M^{\text{res}} x_o^{\text{adv}}. \tag{11}$$

to find x_o^{adv}. Note that for $s_o > s_m$, this is an underdetermined system so there exist multiple solutions. A minimum-norm solution, x_o^*, can be obtained by computing the right pseudo-inverse of M^{res} given by

$$(M^{\text{res}})^+ = (M^{\text{res}})^\top (M^{\text{res}}(M^{\text{res}})^\top)^+ \tag{12}$$

$$x_o^* = (M^{\text{res}})^+ x_m^{\text{adv}} \tag{13}$$

However, the adversary does not want to find a minimum-norm original sample x_o^* but rather a minimum-norm perturbation $\delta_o^* = x_o^{\text{adv}} - x_o$. This can be accomplished by modifying Eqn. (11) and Eqn. (13) slightly

$$M^{\text{res}}\left(x_o + \delta_o^*\right) = x_m^{\text{adv}} \tag{14}$$

$$M^{\text{res}}\delta_o^* = x_m^{\text{adv}} - M^{\text{res}}x_o \tag{15}$$

$$\delta_o^* = (M^{\text{res}})^+ \left(x_m^{\text{adv}} - M^{\text{res}}x_o\right) \tag{16}$$

$$\delta_o^* = (M^{\text{res}})^+ \left(x_m^{\text{adv}} - x_m\right). \tag{17}$$

Eqn. (17) summarizes the recovery phase for resizing. By construction, it guarantees that δ_o^* is a minimum-norm perturbation for a given x_m^{adv}, or $x_o^{\text{adv}} = x_o + \delta_o^*$ is a projection of x_o onto the set of solutions that map to x_m^{adv} after resizing. In other words, by replacing any δ_o with $z_o - x_o$, we have

$$x_o^{\text{adv}} = \underset{z_o \in \mathbb{R}^{s_o \times s_o}}{\arg\min} \quad \|z_o - x_o\|_2 \tag{18}$$

$$\text{s.t.} \quad M^{\text{res}} z_o = x_m^{\text{adv}}. \tag{19}$$

In practice, we can compute δ_o^* by either using an iterative solver on Eqn. (11) directly, or by pre-computing the pseudo-inverse in Eqn. (12). The former does not require caching any matrix but must be recomputed for every input. Caching the pseudo-inverse is more computationally expensive but is done only once. Since M^{res} is sparse, both options are very efficient.

C. Biased-Gradient Attack

C.1. Details on the Recovery Phase

We propose a recovery phase for general preprocessors which should also work for cropping and resizing as well, albeit less efficiently compared to the one in Bypassing Attack. Assuming that the preprocessor is differentiable or has a differentiable approximation, it is possible to replace the exact projection mechanism for finding x_o^{adv} with an iterative method. Specifically, consider relaxing the constraint from Eqn. (1) with a Lagrange multiplier:

$$\underset{z_o \in \mathcal{X}_o}{\arg\min} \quad \|z_o - x_o\|_2^2 + \lambda \left\|t(z_o) - x_m^{\text{adv}}\right\|_2^2. \tag{20}$$

This optimization problem can then be solved with gradient descent combined with a binary search on the Lagrange multiplier λ. We emphasize that, unlike the exact recovery for resizing or cropping, the second term does not necessarily need to be driven down to zero, i.e., $t(z_o^*) = x_m^{\text{adv}}$. For the Biased-Gradient Attack, x_m^{adv} can be seen as a proxy to make z_o^* misclassified by $f(t(\cdot))$ or as a guide to move $t(z_o)$ towards. Specifically, we want the smallest λ such that the solution z_o^* minimizes $\|z_o^* - x_o\|_2$ while also being misclassified.

To this end, we use binary search on λ by increasing/decreasing it when z_o^* is correctly/incorrectly classified. Throughout this paper, we use 10 binary search steps (3 steps in the case of the neural-network-based preprocessor as computing gradients through these models can be expensive). Each step only requires exactly one query to check the predicted label at the end. In practice, we also impose a constraint that keeps z_0 in the input domain $[0, 1]$ using a change of variable trick inspired by the attack from Carlini & Wagner (2017).

D. Preprocessor Extraction Attacks

Algorithm 3 Outline of our extraction attack. Here, we denote the target classification pipeline as $F := f \circ t$ where f and t are unknown to the adversary.

Input: Target classifier API F, a pair of images x_0, x_1 where $F(x_0) \neq F(x_1)$, set of guessed preprocessors $\mathbb{T}$.
Output: Extracted preprocessor t^*
$(u_0, u_1) \leftarrow$ `GenUnstablePair`$(F, (x_0, x_1))$
 # (Optional) Estimate p to compute `num_trials` *given the desired p-value α*
`num_trials` $\leftarrow$ `GetNumTrials`(F, Q, p, α)
for $t \in \mathbb{T}$ **do**
  for $i = 1$ **to** `num_trials` **do**
    $(u_0', u_1') \leftarrow$ `GenPreImage`$(t, (u_0, u_1))$
    $c_i \leftarrow F(u_0') = F(u_0)$ & $F(u_1') = F(u_1)$
  end for
  if $\forall_i c_i = 1$ **then**
    return t
  end if
end for

D.1. Detailed Construction of Unstable Pairs

We begin by identifying (any) two images x_0, x_1 such that $f(t(x_0)) \neq f(t(x_1))$. This step should be easy: it suffices to identify two valid images that actually belong to different classes, or to make random (large-magnitude) modifications to one image x_0 until it switches classes and then call the perturbed image x_1. Intuitively, because $f(t(x_0)) \neq f(t(x_1))$, if we were to interpolate between x_0 and x_1, there must be a midpoint $\bar{x}$ where the decision changes. By picking x_0 and x_1 to straddle this midpoint $\bar{x}$, we obtain an unstable example pair. If the input space of the pipeline were continuous, we can generate an unstable pair, up to the floating-point precision, with a single binary search. However, since we focus on real systems that accept only 8-bit images, we need to take multiple extra steps to create the pair that differs by only one bit on one pixel.

First, we reduce the ℓ_0 difference between the two images via binary search. Construct a new image $\bar{x}$ where each pixel is independently chosen (uniformly at random) as the pixel value either from the image x_0 or from the image x_1. This new image $\bar{x}$ now roughly shares half of the pixels with x_0 and half of the pixels with x_1. If $f(t(\bar{x})) = f(t(x_0))$ replace x_0 with $\bar{x}$ and repeat; if $f(t(\bar{x})) = f(t(x_1))$ then replace x_1 with $\bar{x}$ and repeat.

Next, reduce the ℓ_∞ difference between these two images, again following the same binary search procedure. Let $\bar{x} = (x_0 + x_1)/2$, and query the model to obtain $f(t(\bar{x}))$. If $f(t(\bar{x})) = f(t(x_0))$ then replace x_0 with $\bar{x}$ and repeat; if $f(t(\bar{x})) = f(t(x_1))$ then replace x_1 with $\bar{x}$ and repeat. Do this until x_0 and x_1 differ from each other by at most 1/255 (the smallest difference two images can have). This will eventually give a pair of images that now differ in exactly one pixel coordinate, and in this one coordinate by exactly 1/255. By construction, these two images are also classified as different classes by the pipeline F. We call them an unstable pair. Note that we have not relied on the knowledge of t as we have only treated $f \circ t$ as a single function.

D.2. Detailed Pre-Image Attack

Once we obtain the unstable pair (u_0, u_1), the next step is to use them to generate multiple pre-image (u_0', u_1') and "check" our "guess" of the preprocessor. Before explaining how the pre-images are generated, we will first expand on the implications of the two outcomes: our guess is either right or wrong.

Our guess is correct: In the case that our guess is right, ($\tilde{t} = t^*$), the following equality will hold for $i \in \{0, 1\}$,

$$f(t^*(u_i')) = f(\tilde{t}(u_i')) = f(\tilde{t}(u_i)) = f(t^*(u_i)) \tag{21}$$

where the first equality holds by the assumption that $\tilde{t} = t^*$, the second equality holds by construction that u_i' and u_i are

pre-images, and the final equality holds under the first correctness assumption. From here, we can conclude

$$\underbrace{f(t^*(u_0')) = \overbrace{f(t^*(u_0))}^{\text{By construction}}}_{\text{By Eqn. (21)}} \neq \underbrace{\overbrace{f(t^*(u_1))}^{\text{By construction}} = f(t^*(u_1'))}_{\text{By Eqn. (21)}}.$$

Put simply, this means that if we feed the pipeline with u_0' and u_1', and if our preprocessor guess is correct, then the pipeline will give two different answers $f(t^*(u_0')) \neq f(t^*(u_1'))$.

Our guess is wrong: On the other hand, if our guess at the preprocessor was wrong, i.e., $\tilde{t} \neq t^*$, then we will, with high probability, observe a different outcome:

$$\underbrace{f(t^*(u_0')) = \overbrace{f(t^*(z(u_0)))}^{\text{By definition of an } \textit{unstable example pair}} = \underbrace{f(t^*(z(u_1)))}_{\text{By construction}} = f(t^*(u_1'))}_{\text{By construction}}$$

where the middle inequality holds true because the examples u_0 and u_1 are an unstable example pair, and z is the non-identity transformation used to construct u_i' from u_i.

By coming up with multiple pre-images, querying the target pipeline, and observing the predictions, we can check whether our guess on the preprocessor is correct or not.

D.2.1. A Greedy Pre-image Attack

The previous step requires the ability to construct the pre-images for an arbitrary image x and an arbitrary guessed transformation $\tilde{t}$. While in general, this problem is intractable (e.g., a cryptographic hash function resists exactly this), common image preprocessors are not explicitly designed to be robust and so in practice, it is often nearly trivial.

In practice, we implement this attack via a greedy and naive attack that works well for any transformation that operates over discrete integers $t : \mathbb{Z}^n \to \mathbb{Z}^m$, which is the case for image preprocessors where pixel values lie between 0 and 255.

To begin, let a_0 be the image whose pre-image we would like to compute. We then make random pixel-level perturbations to the image a_0 by randomly choosing a pixel coordinate j and either increasing or decreasing its value by $1/255$. We refer to each of these as $\{a_0^j\}_{j=0}^J$. We take each of these candidates a_0^j and check if $\tilde{t}(a_0^j) = \tilde{t}(a_0)$. If any hold true, then we *accept* this change and let $a_1 = a_0^j$. We then repeat this procedure with a_1 to get a sequence of images $a_0, a_1 \dots a_K$ so that $\tilde{t}(a_0) = \cdots = \tilde{t}(a_K)$ and that $\|a_0 - a_K\|$ is sufficiently large. We desire large perturbation because, intuitively, the larger the difference, the higher the probability that the unstable property will hold. In other words, it is more likely that $f(t(z(u_0'))) = f(t(z(u_1')))$ if $\tilde{t} \neq t$, where u_0' and u_1' are a_K and b_K in this case. In practice, we only use one unstable example pair, but if more confidence is desired, an attacker could use many (at an increased query cost).

E. Additional Experiment Results

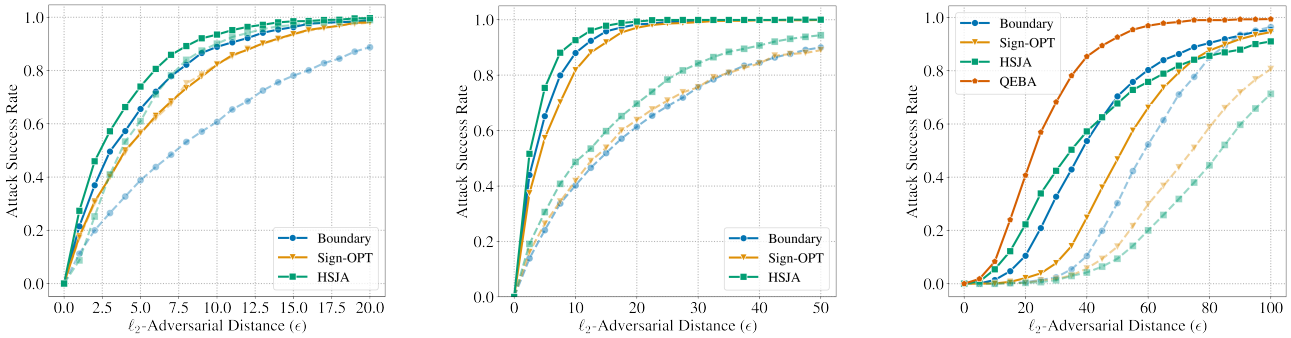
E.1. Complete Preprocessor-Aware Attack Results

Notice that, for the nearest resizing, our Bypassing Attack finds adversarial examples with about the same mean adversarial distance as the no-preprocessor case **regardless of the input dimension** (see Table 7). It may seem counter-intuitive: one might expect that the ℓ_2-norm of the adversarial perturbation scales with the square root of the input dimension. This may be the case if a new classifier were trained on each of the different input sizes (Shafahi et al., 2019). However, this observation matches the intuition that similar to cropping, the nearest resize operation only keeps 224×224 pixels regardless of the input dimension. Hence, only the perturbation on these pixels matters to the prediction.

To build some more intuition on this phenomenon, let's consider a toy example of a binary classifier that simply classifies one-dimensional data, e.g., white and black pixels with values of 0 and 1 respectively, by using a 0.5 threshold. To push a white pixel over the decision boundary (or the threshold, in this case) requires a perturbation of size 0.5. Now consider a new set of inputs with size 2×2 and a nearest resize that maps the 2×2 inputs to one pixel. The classifier remains unchanged. In this case, the nearest resize simply picks one pixel (say, the top left) out of the four pixels. Which pixel is picked depends on the exact implementation but does not matter for our purpose here. To attack this classifier from a 2×2 input, the adversary still needs to change only the top left pixel by 0.5, and thus, the adversarial distance remains unchanged. Even for larger

Table 6: Comparing the mean adversarial perturbation norm (↓) for cropping. The numbers in the parentheses indicate s_o and s_m, respectively. "Δ" is a ratio between the perturbation norm under a preprocessor-unaware ("Unaware") vs our Bypassing Attack, both using their respectively best set of hyperparameters. The smallest adversarial distance found with untargeted and targeted attacks is in bold. For the distance, lower is better.

Preprocessors	Methods	Hparams	Untargeted Attacks			Targeted Attacks			
			Boundary	Sign-OPT	HSJA	Boundary	Sign-OPT	HSJA	QEBA
Crop (256 → 224)	Unaware	Default	11.1	6.7	4.4	48.6	50.6	40.9	24.7
		Best	5.3	6.5	4.2	42.8	50.4	38.2	22.2
	Bypassing (ours)	Default	9.6	5.9	3.9	42.3	46.0	35.1	21.2
		Best	4.6	5.8	**3.6**	37.3	46.0	32.9	**19.6**
		Δ	1.16×	1.12×	1.16×	1.15×	1.08×	1.16×	1.13×



(a) No Preprocessor (b) Resize (1024 → 224, nearest) (c) Quantize (8 bits)

Figure 5: Plots of the attack success rate at varying maximum adversarial distance with different preprocessors. The darker solid lines denote the preprocessor-unaware and the Bypassing attacks with their respectively best hyperparameters. The dashed lines denote the default hyperparameters, and the remaining lighter solid lines correspond to the other set of hyperparameters we sweep.

input sizes, only one pixel will still be selected. While this toy example explains resizing with the nearest interpolation, it does not necessarily apply to bilinear or bicubic. Nonetheless, all of our experimental results support this hypothesis.

E.2. Attack Hyperparameter Choices

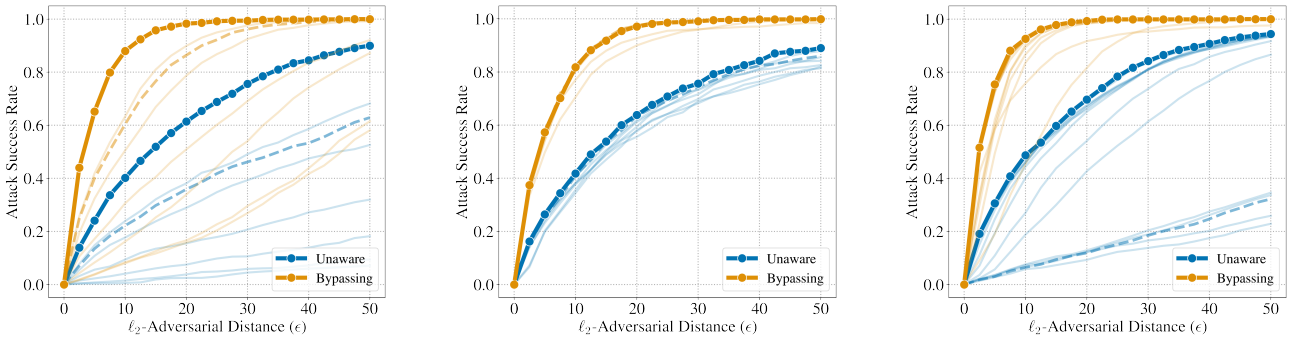
We have seen from Section 4 that fine-tuning the hyperparameters improves the attack significantly in most cases. We discuss when it is most important for the adversary to fine-tune their attack hyperparameters. Figure 6 (Appendix E) shows the attack success rate at varying adversarial distances for three untargeted attack algorithms. For Boundary, HSJA, and QEBA attacks, the gain from selecting the right set of hyperparameters is significant, a large improvement over the default.

For instance, a properly tuned Boundary attack outperforms Sign-OPT and HSJA attacks with their default hyperparameters in majority of the settings with resizing preprocessor.

For most attacks, we do not observe a universally good set of hyperparameters across different preprocessors. However, there are two general rules of thumb when it comes to better guess the hyperparameters:

1. Using a larger value of γ (10^3–10^4) in HSJA attack is almost always better than the default (10). This applies to both preprocessor-aware and -unaware attacks and to all preprocessors.

2. QEBA attack samples the noise used for gradient approximation from an image space with a smaller size $rs_o \times rs_o$ where s_o is the original input size, and r is the hyperparameter smaller than 1. The default value of r is $\frac{1}{4}$ for $s_o = 224$. Consequently, for a larger s_o such as the resizing preprocessor, setting r to be smaller accordingly is always beneficial. For example, we find that for $s_o = 256, 512, 1024$, the best values of r are $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, respectively.



(a) Boundary Attack (b) Sign-OPT Attack (c) HSJA

Figure 6: Plots of the attack success rate at varying maximum adversarial distance. Here, the preprocessor is resizing with the nearest interpolation from 1024×1024 to 224×224, corresponding to the first five rows of Table 7 (untargeted). The solid lines with markers denote the preprocessor-unaware and the Bypassing attacks with their respective best hyperparameters. The dashed lines denote the default hyperparameters, and the remaining lighter solid lines correspond to the other set of hyperparameters we sweep.

Here, we include two figures that compare the effect of tuning the attack hyperparameters in multiple settings. Figure 5 suggests that the default hyperparameters often work well as expected when no preprocessor is used while there is much greater discrepancy between the default and the best hyperparameters when preprocessors are used.

The degree in which the hyperparameter tuning matters also depends on the attack algorithm. Figure 6 visually compares the effectiveness of three untargeted attacks on the resizing preprocessor. It is obvious that Boundary and HSJA attacks benefit much more from a hyperparameter sweep compared to Sign-OPT attack.

For the numerical comparison between the best and the default hyperparameter choices on quantization and JPEG, please refer to Table 9 and Table 8, respectively. Tables 10 to 12 show the results with all the hyperparameters we sweep for resizing, cropping, and quantization/JPEG, respectively. In the most extreme case, the best hyperparameters reduce the mean adversarial distance by a factor of $15\times$ for JPEG with quality 60 under untargeted HSJA + our Biased-Gradient attack.

E.3. Varying Number of Attack Iterations

There are two interesting properties we observe when we vary the number of queries the adversary can utilize. So far we have considered attacks that use exactly 5,000 queries; in this section, we now test attacks with 500 to 50,000 queries. Figure 4 plots the mean adversarial distance as a function of the number of queries for QEBA attack with the best hyperparameter for each respective setting. First, the adversarial distance plateaus after around 10,000 queries, and the distance found by preprocessor-unaware attacks never reaches that of Bypassing/Biased-Gradient Attack. This suggests that our preprocessor-aware attack does not only improve the efficiency of the attack algorithms but also allows it to find closer adversarial examples that would have been completely missed otherwise.

The second observation is that the improvement from Bypassing Attack over the preprocessor-unaware attack is consistent across all numbers of queries. For instance, in Figure 4a, the Bypassing Attack reduces the mean adversarial distance by a factor of around 4.5 to 4.8 for any number of queries. This is not the case for the Biased-Gradient Attack which is relatively more effective at a larger number of queries. In Figure 4b, Biased-Gradient Attack yields an improvement of $1.1\times$ at 500 queries and $2.5\times$ beyond 10,000 queries.

Figure 7: Randomly selected test images including the original and the adversarial. Our Biased-Gradient attack finds adversarial examples with a noticeably less perceptible perturbation compared to the ones generated by the preprocessor-unaware baseline. The target preprocessor here is 8-bit quantization, and the base attack is targeted QEBA. Looking closely, one can see some detail of the initialized images from the target class left in the adversarial images.

Table 7: Comparing the mean adversarial perturbation norm for resizing with different interpolation methods and input sizes. We report only the best hyperparameter found after the sweep. Lower is better.

Preprocessors	Methods	Untargeted Attacks			Targeted Attacks			
		Boundary	Sign-OPT	HSJA	Boundary	Sign-OPT	HSJA	QEBA
Resize ($1024 \rightarrow 224$) (Nearest)	Unaware	21.2	24.8	16.5	172.2	198.8	153.4	90.5
	SNS	n/a	n/a	3.9	n/a	n/a	112.6	32.2
	Bypassing (ours)	4.7	5.8	**3.7**	37.7	46.3	33.3	**19.4**
	Biased-Grad (ours)	n/a	n/a	3.6	n/a	n/a	32.9	19.6
Resize ($512 \rightarrow 224$) (Nearest)	Unaware	10.3	12.5	8.1	84.7	97.8	74.2	44.5
	SNS	n/a	n/a	3.7	n/a	n/a	56.5	54.1
	Bypassing (ours)	4.5	5.7	**3.6**	37.3	45.5	32.6	**19.4**
	Biased-Grad (ours)	n/a	n/a	3.6	n/a	n/a	34.2	19.9
Resize ($256 \rightarrow 224$) (Nearest)	Unaware	6.3	6.1	3.9	41.0	50.6	36.1	20.1
	SNS	n/a	n/a	3.4	n/a	n/a	34.5	30.0
	Bypassing (ours)	7.7	5.4	3.4	36.0	44.8	31.3	17.9
	Biased-Grad (ours)	n/a	n/a	**3.4**	n/a	n/a	31.4	**17.6**
Resize ($1024 \rightarrow 224$) (Bilinear)	Unaware	32.7	38.2	25.5	198.3	213.0	188.4	90.3
	SNS	n/a	n/a	5.7	n/a	n/a	113.7	111.6
	Bypassing (ours)	7.4	9.1	6.0	58.2	70.9	50.3	**30.0**
	Biased-Grad (ours)	n/a	n/a	**5.6**	n/a	n/a	56.4	36.4
Resize ($512 \rightarrow 224$) (Bilinear)	Unaware	15.9	19.1	12.6	98.7	106.0	90.8	45.6
	SNS	n/a	n/a	**5.5**	n/a	n/a	65.1	57.0
	Bypassing (ours)	7.4	9.2	5.9	57.7	70.9	50.2	**30.3**
	Biased-Grad (ours)	n/a	n/a	5.5	n/a	n/a	51.1	30.5
Resize ($256 \rightarrow 224$) (Bilinear)	Unaware	6.3	7.8	5.1	45.6	53.0	40.8	21.9
	SNS	n/a	n/a	3.6	n/a	n/a	33.9	29.0
	Bypassing (ours)	7.7	9.9	6.1	45.5	57.8	46.2	**21.5**
	Biased-Grad (ours)	n/a	n/a	**3.4**	n/a	n/a	34.0	22.0
Resize ($1024 \rightarrow 224$) (Bicubic)	Unaware	25.7	29.2	20.6	184.8	207.3	171.6	91.2
	SNS	n/a	n/a	4.5	n/a	n/a	108.3	55.7
	Bypassing (ours)	5.8	7.1	**4.5**	46.4	57.7	40.6	**23.8**
	Biased-Grad (ours)	n/a	n/a	4.6	n/a	n/a	45.8	28.1
Resize ($512 \rightarrow 224$) (Bicubic)	Unaware	13.1	15.4	10.1	91.1	101.5	81.1	44.3
	SNS	n/a	n/a	4.6	n/a	n/a	59.9	55.8
	Bypassing (ours)	5.8	7.0	**4.5**	46.4	56.6	40.2	**24.4**
	Biased-Grad (ours)	n/a	n/a	4.5	n/a	n/a	42.6	25.6
Resize ($256 \rightarrow 224$) (Bicubic)	Unaware	6.0	7.4	4.8	44.2	51.9	39.4	21.5
	SNS	n/a	n/a	4.0	n/a	n/a	74.9	87.6
	Bypassing (ours)	5.8	7.3	4.6	42.5	52.9	37.6	21.6
	Biased-Grad (ours)	n/a	n/a	**3.9**	n/a	n/a	36.4	**21.2**

Table 8: Comparison of the mean adversarial perturbation norm (↓) for JPEG compression among the baseline attack unaware of the preprocessor, SNS, and our Biased-Gradient Attack. "Δ" is a ratio between the perturbation norm under a preprocessor-unaware ("Unaware") vs our Bypassing Attack, both using their respectively best set of hyperparameters.

Preprocess	Methods	Attack Hyperparameter	Untargeted	Targeted	
			HSJA	HSJA	QEBA
JPEG (quality 100)	Unaware	Default	5.7	35.8	18.8
		Best	3.5	31.9	**18.8**
	SNS	Default	3.1	43.0	24.7
		Best	2.4	**29.5**	24.7
	Biased-Grad (ours)	Default	28.9	71.9	19.2
		Best	**2.8**	32.5	19.2
		Δ	1.23×	0.98×	0.98×
JPEG (quality 80)	Unaware	Default	29.6	85.7	50.7
		Best	8.9	63.2	43.9
	SNS	Default	13.2	64.3	47.2
		Best	7.3	45.4	47.2
	Biased-Grad (ours)	Default	23.7	80.4	25.5
		Best	**2.3**	**29.2**	**21.9**
		Δ	3.87×	2.16×	2.00×
JPEG (quality 60)	Unaware	Default	29.2	86.8	56.1
		Best	9.2	63.2	52.7
	SNS	Default	11.9	66.1	44.6
		Best	2.7	44.5	44.6
	Biased-Grad (ours)	Default	22.2	82.0	27.0
		Best	**1.5**	**25.1**	**26.1**
		Δ	6.13×	2.51×	2.02×

Table 9: Comparison of the mean adversarial perturbation norm (↓) for quantization among the baseline attack unaware of the preprocessor, SNS, and our Biased-Gradient Attack. "Δ" is a ratio between the perturbation norm under a preprocessor-unaware ("Unaware") vs our Bypassing Attack, both using their respectively best set of hyperparameters.

Preprocess	Methods	Attack Hyperparameter	Untargeted	Targeted	
			HSJA	HSJA	QEBA
Quantize (8 bits)	Unaware	Default	29.1	83.6	26.5
		Best	5.0	45.6	26.5
	SNS	Default	6.8	42.8	35.0
		Best	4.6	42.8	35.0
	Biased-Grad (ours)	Default	7.1	46.2	21.3
		Best	**3.9**	**33.9**	**20.6**
		Δ	1.27×	1.35×	1.29×
Quantize (6 bits)	Unaware	Default	30.4	86.1	40.6
		Best	7.5	48.2	39.4
	SNS	Default	17.5	76.2	43.7
		Best	5.9	46.8	43.7
	Biased-Grad (ours)	Default	11.1	56.7	25.1
		Best	**3.9**	**34.2**	**23.3**
		Δ	1.92×	1.41×	1.69×
Quantize (4 bits)	Unaware	Default	32.3	88.9	58.4
		Best	9.7	63.7	56.4
	SNS	Default	22.2	76.5	57.2
		Best	6.4	55.9	57.2
	Biased-Grad (ours)	Default	19.2	74.7	31.8
		Best	**3.1**	**39.3**	**28.8**
		Δ	3.05×	1.54×	1.86×

Table 10: Comparison of the mean adversarial perturbation norm (↓) for the resizing preprocessor with nearest interpolation (1024 → 224 pixels) among the baseline attack unaware of the preprocessor, SNS, and our Bypassing Attack under various hyperparameters of each attack algorithm. Notice the generally large gap between the best hyperparameter (**bold**) vs the default one (first row for each respective attack). The underlined number is best for the given attack algorithm. $s_\parallel$ and $s_\perp$ denote the Boundary attack's step size in the parallel and the orthogonal direction to a given input, respectively. Note that SNS is only applicable to HSJA and QEBA attacks, and for QEBA, SNS replaces its subspace noise sampling process. Hence, we only report a single number for SNS + QEBA attack.

Preprocessor	Attack	Attack Parameters	Unaware	SNS	Ours
Resize (nearest, 1024 → 224)	Boundary (untargeted)	$s_\parallel = 0.01, s_\perp = 0.01$	45.4		9.8
		$s_\parallel = 0.1, s_\perp = 0.01$	56.1		12.4
		$s_\parallel = 0.001, s_\perp = 0.01$	134.1		29.1
		$s_\parallel = 0.01, s_\perp = 0.1$	**21.2**		**4.7**
		$s_\parallel = 0.01, s_\perp = 0.001$	40.6		8.8
	Sign-OPT (untargeted)	$\alpha = 0.2, \beta = 0.001$	**24.8**		5.8
		$\alpha = 2, \beta = 0.001$	25.2		5.9
		$\alpha = 0.02, \beta = 0.001$	25.0		**5.8**
		$\alpha = 0.2, \beta = 0.01$	25.3		5.8
		$\alpha = 0.2, \beta = 0.0001$	25.4		5.9
	HSJA (untargeted)	$\gamma = 10^1$	28.5		3.8
		$\gamma = 10^2$	18.3		**3.7**
		$\gamma = 10^3$	17.4	12.4	3.7
		$\gamma = 10^4$	16.8	4.8	3.8
		$\gamma = 10^5$	**16.5**	**3.9**	8.0
		$\gamma = 10^6$	26.2	6.1	60.6
	Boundary (targeted)	$s_\parallel = 0.01, s_\perp = 0.01$	194.4		42.3
		$s_\parallel = 0.1, s_\perp = 0.01$	242.6		52.4
		$s_\parallel = 0.001, s_\perp = 0.01$	310.2		67.8
		$s_\parallel = 0.01, s_\perp = 0.1$	233.1		50.6
		$s_\parallel = 0.01, s_\perp = 0.001$	**172.2**		**37.7**
	Sign-OPT (targeted)	$\alpha = 0.2, \beta = 0.001$	201.3		**46.3**
		$\alpha = 2, \beta = 0.001$	**199.4**		46.4
		$\alpha = 0.02, \beta = 0.001$	200.2		46.4
		$\alpha = 0.2, \beta = 0.01$	203.3		47.1
		$\alpha = 0.2, \beta = 0.0001$	202.4		46.4
	HSJA (targeted)	$\gamma = 10^1$	168.3		35.2
		$\gamma = 10^2$	160.5		34.0
		$\gamma = 10^3$	159.7	122.7	33.3
		$\gamma = 10^4$	**153.4**	**112.6**	**23.5**
		$\gamma = 10^5$	162.0	212.5	37.3
	QEBA (targeted)	Naive, $\gamma = 0.01$	138.7	**32.2**	29.7
		Resize 2×,$\gamma = 0.01$	139.1		21.9
		Resize 4×,$\gamma = 0.01$	124.5		**19.4**
		Resize 8×,$\gamma = 0.01$	103.7		19.9
		Resize 16×,$\gamma = 0.01$	92.5		26.3
		Resize 32×,$\gamma = 0.01$	**90.5**		42.9

Table 11: Comparison of the mean adversarial perturbation norm (↓) for the cropping preprocessor (256 → 224 pixels). For the full description, please refer to Table 11.

Preprocessor	Attack	Attack Parameters	Unaware	SNS	Ours
Crop (256 → 224)	Boundary (untargeted)	$s_\parallel = 0.01, s_\perp = 0.01$	11.1		9.6
		$s_\parallel = 0.1, s_\perp = 0.01$	**5.3**		**4.6**
		$s_\parallel = 0.001, s_\perp = 0.01$	32.8		28.5
		$s_\parallel = 0.01, s_\perp = 0.1$	14.1		12.1
		$s_\parallel = 0.01, s_\perp = 0.001$	10.1		8.7
	Sign-OPT (untargeted)	$\alpha = 0.2, \beta = 0.001$	6.7		5.9
		$\alpha = 2, \beta = 0.001$	6.8		5.9
		$\alpha = 0.02, \beta = 0.001$	6.6		**5.8**
		$\alpha = 0.2, \beta = 0.01$	**6.5**		5.9
		$\alpha = 0.2, \beta = 0.0001$	6.7		5.9
	HSJA (untargeted)	$\gamma = 10^2$	4.4	3.7	3.6
		$\gamma = 10^3$	**4.2**	**3.7**	**3.6**
		$\gamma = 10^4$	4.2	3.8	4.9
		$\gamma = 10^5$	6.0	6.9	4.9
	Boundary (targeted)	$s_\parallel = 0.01, s_\perp = 0.01$	48.6		42.3
		$s_\parallel = 0.1, s_\perp = 0.01$	62.0		53.1
		$s_\parallel = 0.001, s_\perp = 0.01$	76.9		66.8
		$s_\parallel = 0.01, s_\perp = 0.1$	58.0		50.5
		$s_\parallel = 0.01, s_\perp = 0.001$	**42.8**		**37.3**
	Sign-OPT (targeted)	$\alpha = 0.2, \beta = 0.001$	52.8		**46.3**
		$\alpha = 2, \beta = 0.001$	**52.7**		46.5
		$\alpha = 0.02, \beta = 0.001$	52.9		46.4
		$\alpha = 0.2, \beta = 0.01$	53.9		47.7
		$\alpha = 0.2, \beta = 0.0001$	53.2		46.7
	HSJA (targeted)	$\gamma = 10^1$	40.4		34.9
		$\gamma = 10^2$	38.7	36.4	33.6
		$\gamma = 10^3$	**38.2**	**35.4**	**32.9**
		$\gamma = 10^4$	47.4	46.3	44.7
		$\gamma = 10^5$	104.5	104.2	92.9
	QEBA (targeted)	Naive, $\gamma = 0.01$	34.0	**31.5**	29.5
		Resize 2×,$\gamma = 0.01$	24.7		21.2
		Resize 4×,$\gamma = 0.01$	**22.2**		**19.6**
		Resize 8×,$\gamma = 0.01$	23.2		20.3
		Resize 16×,$\gamma = 0.01$	30.0		26.8

Table 12: Comparison of the mean adversarial perturbation norm (↓) for the quantization preprocessor (4 bits) and JPEG compression (quality of 60). For the full description, please refer to Table 11.

Preprocessor	Attack	Attack Parameters	Unaware	SNS	Ours
Quantize (4 bits)	HSJA (untargeted)	$\gamma = 10^2$	29.8	22.2	4.4
		$\gamma = 10^3$	22.7	12.3	**3.1**
		$\gamma = 10^4$	11.2	**6.4**	10.3
		$\gamma = 10^5$	**9.7**	123.3	42.6
	HSJA (targeted)	$\gamma = 10^2$	85.3	76.5	54.5
		$\gamma = 10^3$	74.7	57.6	**39.3**
		$\gamma = 10^4$	**63.7**	**55.9**	48.8
		$\gamma = 10^5$	95.2	92.1	91.3
	QEBA (targeted)	Naive, $\gamma = 0.01$	71.8	**57.2**	33.6
		Resize 2×,$\gamma = 0.01$	61.2		31.7
		Resize 4×,$\gamma = 0.01$	58.4		30.4
		Resize 8×,$\gamma = 0.01$	**56.4**		**28.8**
		Resize 16×,$\gamma = 0.01$	60.4		29.2
JPEG (quality 60)	HSJA (untargeted)	$\gamma = 10^2$	27.5	11.9	10.2
		$\gamma = 10^3$	21.3	6.6	5.0
		$\gamma = 10^4$	10.5	**2.7**	2.0
		$\gamma = 10^5$	**9.2**	3.4	**1.5**
	HSJA (targeted)	$\gamma = 10^2$	80.5	66.1	66.1
		$\gamma = 10^3$	66.4	**44.5**	43.5
		$\gamma = 10^4$	**63.2**	51.8	**25.1**
		$\gamma = 10^5$	93.3		93.5
	QEBA (targeted)	Naive, $\gamma = 0.01$	64.9	**44.6**	28.3
		Resize 2×,$\gamma = 0.01$	58.5		21.1
		Resize 4×,$\gamma = 0.01$	56.1		**21.0**
		Resize 8×,$\gamma = 0.01$	**52.7**		22.7
		Resize 16×,$\gamma = 0.01$	53.3		25.9