Boosting Universal Adversarial Attack on Deep Neural Networks

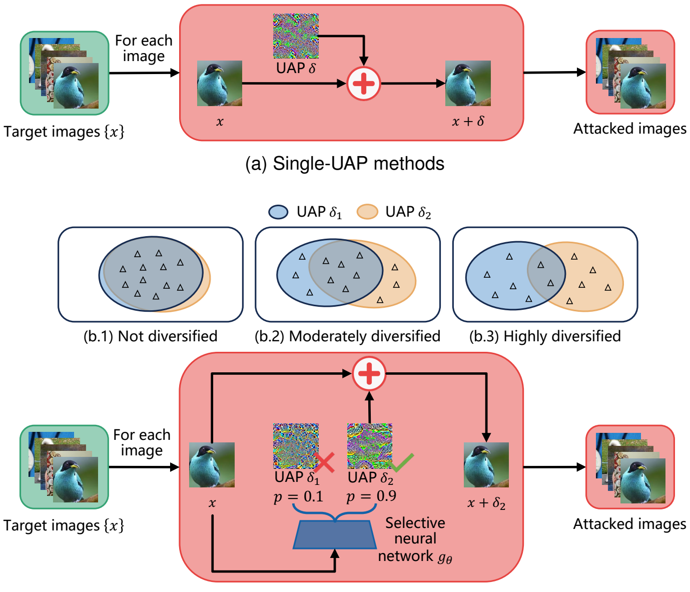
Shaoxin Li, Xiaofeng Liao, *Fellow, IEEE,*, Xin Che, and Lingyang Chu

***Abstract*—Deep neural networks (DNNs) are well-known to be susceptible to many universal adversarial perturbations (UAPs), where each UAP can successfully attack many images when added to the input. In this paper, we explore the existence of diversified UAPs, each of which successfully attacks a large but substantially different set of images. Since the sets of images successfully attacked by different UAPs are often complementary to each other, strategically selecting the most effective UAP to attack each new image could maximize the overall coverage of successful attacks. Following this insight, we propose a novel attack framework named boosting universal adversarial attack. The key idea is to simultaneously train a set of diversified UAPs and a selective neural network, such that the selective neural network can choose the most effective UAP when attacking a new target image. Due to the simplicity and effectiveness of the proposed boosting attack framework, it can be generally used to significantly boost the attack effectiveness of many classic single-UAP methods that only use a single UAP to attack all target images. Meanwhile, the boosting attack framework is also able to perform real-time attacks as it does not require any additional training or fine-tuning when attacking new target images. Extensive experiments demonstrate the outstanding performance of the proposed boosting attack framework.**

***Index Terms*—Universal adversarial perturbation, deep neural network, boosting attack.**

I. Introduction

DEEP neural networks (DNNs) have achieved notable milestones in various computer vision tasks [2]–[5]. Despite their success, DNNs are known to be susceptible to *universal adversarial perturbation (UAP)* [6] – a single carefully crafted, imperceptible perturbation [7]–[12] that, when added to target images, can successfully attack the majority of them by altering a victim DNN's predictions on them. For example, many studies [6], [13]–[17] have proposed to generate a UAP to make image classifiers produce erroneous classification results. As another example, the same idea has also been adopted by [18]–[20] to generate universal adversarial patches against aerial detectors.



(a) Single-UAP methods

(b.1) Not diversified (b.2) Moderately diversified (b.3) Highly diversified

(b) Boosting universal adversarial attack method when $K = 2$

Fig. 1. The overview diagrams of single-UAP methods and the proposed boosting universal adversarial attack method. (a) The single-UAP methods attack each target image by adding the same UAP δ. (b) The boosting attack method attacks each target image via adding the UAP chosen from a set of K diversified UAPs by the selective neural network g_θ. The three Venn diagrams in (b.1), (b.2) and (b.3) illustrate the sets of target images that can be successfully attacked by the UAPs δ_1 and δ_2 when they are not diversified, moderately diversified and highly diversified, respectively. Each elliptical region in these Venn diagrams represents a set of target images in the original image space that a UAP can successfully attack. Each triangle represents a target image in the set.

As demonstrated by many prior works [13]–[17], [21], a UAP not only generalizes well in attacking different target images [13], [16], [21] but also transfers well in attacking different victim DNNs [14], [15], [17]. Consequently, many classic *single-UAP methods* [6], [13]–[17], [21]–[25] train a single UAP and then directly apply it to attack new target images during the attack stage, as shown in Fig. 1(a). This enables extremely fast adversarial attacks and poses a great potential threat to the security of DNNs.

However, it is often challenging to generate a single UAP that can successfully attack a wide variety of target images. Due to the large diversity of image features and the nonlinearity nature of DNNs, a UAP that succeeds in attacking one group of images may fail when attacking many other images [22], [24]. As a result, the single-UAP methods [6],

This work was done by Shaoxin Li during his visit at McMaster University when supervised by Lingyang Chu. This work is an extension of our previously published conference paper [1]. This work is supported in part by the NSERC Discovery Grant program (RGPIN-2022-04977), in part by the Natural Science Foundation of Chongqing (Innovation and Development Joint Fund) under grant CSTB2023NSCQ-LZX0149, in part by the Fundamental Research Funds for the Central Universities under grant 2023CDJKYJH019, and in part by the scholarship from China Scholarship Council. (*Corresponding author: Xiaofeng Liao*)

S. Li and X. Liao are with the College of Computer Science, Chongqing University, Chongqing, 400044, China (email: {shaoxin.li, xfliao}@cqu.edu.cn).

X. Che and L. Chu are with the Department of Computing and Software, McMaster University, Hamilton, L8S 4L8, Canada (email: {chex5, chul9}@mcmaster.ca).

This work has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the authors. The material includes the appendices on additional experimental results of the proposed methods. This material is 3.4 MB in size.

[13]–[17], [21]–[25] that apply a single UAP to all target images generally struggle to achieve high attack effectiveness, especially when the magnitude (e.g., infinity norm) of the UAP is limited to be small by the constraint of imperceptibility.

On the other hand, some prior studies [6], [22], [24] have demonstrated that DNNs are susceptible to not only a single but a large number of UAPs. This indicates the existence of many *diversified* UAPs, each capable of successfully attacking a large set of images, with these sets often being substantially different and complementary to each other. Therefore, strategically selecting the most effective UAP to attack each new target image could maximize the overall attack coverage by combining the strengths of individual UAPs. Such an approach may significantly boost attack effectiveness and thus pose an even greater threat to the security of DNNs.

To verify and study the above security threat, we propose a novel attack framework named **boosting universal adversarial attack**, or **boosting attack** for short. As shown in Fig. 1(b), the *key idea* is to find a set of K diversified UAPs and attack each new target image by the UAP from the set that is most effective in attacking it. For each attack, the most effective UAP is selected by a **selective neural network** that is trained simultaneously with the K diversified UAPs. As such, if each of the K UAPs can successfully attack a substantially different set of images, then the boosting attack framework will significantly improve attack effectiveness because it can successfully attack the union of the images that are successfully attacked by each of the K UAPs. This is illustrated by the Venn diagrams in Figs. 1(b.2) and 1(b.3). Meanwhile, such a boosting attack can also be conducted in real-time because compared to classic single-UAP methods, the only additional overhead of performing a boosting attack is a forward pass of the target image through the selective neural network, which only takes about 2.4 milliseconds on an NVIDIA RTX 3060 GPU.

To the best of our knowledge, the boosting attack framework is a novel universal adversarial attack framework that has not been systematically studied in the literature. As to be discussed in Section II, most existing UAP-based attack methods [6], [13]–[17], [21]–[25] focus on finding a single UAP that successfully attacks the largest group of images. As a result, they pay less attention to finding and utilizing multiple diversified UAPs, thus missing the opportunity to harness the power of boosting universal adversarial attacks.

In this paper, we propose a novel boosting universal adversarial attack framework against DNNs, which significantly enhances the attack effectiveness of single-UAP methods while maintaining a fast attack speed in milliseconds. We make the following contributions.

First, in order to obtain a set of K diversified UAPs, we define the problem of **finding diversified universal adversarial perturbations (FDUAP)** as a clustering problem. The goal of the FDUAP problem is to identify K clusters of images while generating the corresponding UAP for each of the K clusters, such that the number of successfully attacked images in each cluster is maximized.

Second, we propose a novel attack method called **homogeneous boosting attack (HoBA)** to address the FDUAP problem. In HoBA, by introducing a selective neural network to partition images into K clusters, we formulate the FDUAP problem as an optimization problem, where the K diversified UAPs are trained by minimizing the same loss function for maximum attack effectiveness. Then, we solve this optimization problem by a gradient-based method to simultaneously train the K UAPs and the selective neural network in an end-to-end manner. During the attack stage, the trained selective neural network can efficiently select the most effective UAP from the trained K UAPs to attack each new target image, thus enabling both effective and fast boosting attacks.

Third, to further improve the effectiveness of the boosting attack, we extend HoBA to an advanced version called **heterogeneous boosting attack (HeBA)**. HeBA incorporates different types of loss functions to train the set of diversified UAPs, such that the UAPs trained by minimizing different loss functions are more diversified. To achieve this, we extend the optimization problem of HoBA to a new optimization problem and efficiently solve it by proposing a new training method. Due to the enhanced diversity of the UAPs trained by HeBA, the sets of images successfully attacked by these UAPs are more distinct. They have less overlap and collectively cover a wider range of images, as illustrated by the comparison of the Venn diagrams in Figs. 1(b.2) and 1(b.3), and thus could offer higher attack effectiveness.

Last, we conduct extensive experiments to evaluate the attack performance of the proposed boosting attack framework. The experimental results indicate that both HoBA and HeBA significantly improve the attack effectiveness of single-UAP methods while maintaining a fast attack speed in milliseconds, which demonstrates the superiority of the boosting attack framework. Our source code is at https://github.com/ShaoxinLi/Boosting-Universal-Adversarial-Attack.

II. Related Works

To the best of our knowledge, how to conduct boosting universal adversarial attacks on deep neural networks (DNNs) is a novel problem that has not been systematically studied in the literature. It is broadly related to the following two categories of existing adversarial attacks on DNNs.

Image-dependent attacks. Given a target image, image-dependent attacks [26]–[29] perturb the target image by adding a specially tailored perturbation to it, such that DNNs will make an incorrect prediction on the perturbed image. In particular, two subcategories of image-dependent attacks have been studied. The first one is *optimization-based methods* [26], [27], which generate a perturbation for every target image by solving an optimization problem. As a result, they tend to be computationally expensive and cannot achieve fast attacks on new target images [24], [30]. The second one is *generator-based methods* [28], [29], which train a DNN-based generator that maps a target image into a unique perturbation to attack the target image. Since generating a perturbation only involves a forward pass of the target image through the generator, these methods often achieve a much faster attack speed than the optimization-based methods.

Image-agnostic attacks. Instead of crafting a unique perturbation for each target image, the image-agnostic attacks use a universal adversarial perturbation (UAP) to conduct fast attacks. This is achieved by training the UAP to successfully attack as many images as possible and then directly adding the trained UAP to each new target image to launch attacks.

The existence of UAP was first revealed in [6]. Afterward, many UAP-based attack methods were developed for different application contexts. The gradient-based methods [13], [16], [17], [21], [23]–[25] directly optimize a UAP by stochastic gradients to achieve superior attack effectiveness. The class-discriminative methods [30], [31] attempt to craft a UAP specific to images of a chosen group of classes, while having limited influence on the other classes. The GAN-based methods [22], [29] implicitly model the distribution of UAPs by training a generative adversarial network (GAN) [32]. The data-free methods [14], [24], [25] attempt to generate a UAP without using any training images or using only artificial images. The ensemble-based methods [15], [33] focus on improving the across-model transferability of UAP by training a UAP to simultaneously attack multiple DNNs.

Despite the variety of the above UAP-based attack methods, they can all be characterized as *single-UAP methods*, which perform fast attacks by using a single UAP to attack all target images. However, the number of images that can be successfully attacked by a single UAP is often limited, which reduces the attack effectiveness of these methods [24].

Different from the existing works, we develop a novel boosting attack framework for universal adversarial attacks. The key idea is to find a set of K diversified UAPs and attack each new target image with the most effective UAP chosen by a well-trained selective neural network. As demonstrated by the extensive experiments in Section VI, the boosting attack framework significantly improves the attack effectiveness of many classic single-UAP methods [14], [21], [22], [24], [30], and the resulting boosting attack methods even achieve superior performance than the image-dependent generator-based methods [28], [29]. Meanwhile, a boosting attack can also be conducted in milliseconds since the only extra overhead compared to the single-UAP methods is a forward pass of the target image through the selective neural network.

III. The Problem of Finding Diversified UAPs

In this section, we introduce the problem of finding diversified universal adversarial perturbations. Notations frequently used throughout this paper are summarized in Table I.

Following the settings of prior studies [6], [13]–[17], [21]–[25], we focus on the representative image classification task of DNNs in this work. Denote by $X = \{x_i\}_{i=1}^{N}$ a set of N training images with a set of class labels denoted as $\mathcal{C} = \{1, 2, \ldots, C\}$, the victim DNN model to be attacked, denoted by f, is trained on X to perform image classification.

For a target image x, we denote the class label of x predicted by f as $l_f(x) \in \mathcal{C}$. Denote by δ a universal adversarial perturbation (UAP) with the same dimension as x, an attacker performs an attack by adding δ to x. If $l_f(x) \neq l_f(x+\delta)$, then we say x is **successfully attacked** by δ. We call a UAP in short as a **perturbation** when the context is clear.

Now, we define the problem of finding diversified UAPs.

Definition 1. *Given a victim DNN f to be attacked, the training dataset $X = \{x_i\}_{i=1}^{N}$ used to train f, and a real-valued perturbation magnitude $\xi > 0$. The problem of* ***finding diversified universal adversarial perturbations (FDUAP)*** *is to find a set of K perturbations, denoted by $P = \{\delta_1, \ldots, \delta_K\}$, and a set of K non-overlapping clusters of images in X, denoted by $\mathcal{S} = \{S_1, \ldots, S_K\}$, such that*

1) *$S_1 \cup \ldots \cup S_K = X$ and $S_i \cap S_j = \emptyset$ when $i \neq j$;*
2) *in each cluster $S_i \in \mathcal{S}$, the number of images successfully attacked by $\delta_i \in P$ is maximized; and*
3) *for each $\delta_i \in P$, $\|\delta_i\|_\infty \leq \xi$.*

TABLE I
The frequently used notations and their descriptions.

Notation	Description
x	A training image.
X	The set of training images.
N	The number of training images in X.
$\mathcal{C}$	The set of image class labels.
f	The victim DNN model.
$l_f(x)$	The class label of x predicted by f.
δ	A universal adversarial perturbation.
P	The set of universal adversarial perturbations.
K	The number of universal adversarial perturbations in P.
ξ	The real-valued perturbation magnitude.
S	A cluster of images.
$\mathcal{S}$	The set of non-overlapping clusters of images.
$L(x, \delta)$	An attack function used to train perturbations.
$\mathcal{A}$	The set of attack functions.
H	The number of attack functions in $\mathcal{A}$.
R	The number of perturbations trained by each attack function in $\mathcal{A}$.
g_θ	The selective neural network parameterized by θ.
$\mathbb{P}_\theta(x)$	The K-dimensional probability vector output by g_θ.
E	The number of training epochs.
I	The total number of training iterations.
B	The batch size.
t_g	The time cost of computing $\mathbb{P}_\theta(x)$ using g_θ.
t_l	The time cost of computing the value of $L(x_i, \delta_j)$.
t_θ	The time cost of updating θ by the ADAM optimizer.
d	The number of parameters (i.e., pixels) of each perturbation.

The key idea of the above FDUAP problem is to find K clusters of images while generating the corresponding UAP for each of the clusters, such that the number of successfully attacked images in each cluster is maximized. The constraint $\|\delta_i\|_\infty \leq \xi$ limits the infinity norm of δ_i by a small perturbation magnitude ξ such that δ_i is visually imperceptible to humans [22], [30]. Since each perturbation is trained to attack a different cluster of images, the sets of images successfully attacked by different perturbations are likely to be different. Therefore, we regard these perturbations as *diversified*.

In the following two sections, we propose two different methods to solve the FDUAP problem. The first method named **homogeneous boosting attack (HoBA)** uses the same loss function to train the perturbations in P; and the second method named **heterogeneous boosting attack (HeBA)** enhances the diversity of perturbations in P by using different types of loss functions together to train the perturbations in P.

We implement both HoBA and HeBA in the *white-box* setting and the *black-box* setting, respectively. The white-box setting allows an attacker to access the model architecture and parameters of the victim DNN f. The black-box setting adopts the setting of *transfer adversarial attacks* in the literature [16], [17], [22], [30], which only allows the attacker to use f as a black box without knowing its model architecture or parameters. Following the routine of transfer adversarial attacks [16],

Algorithm 1: Solving the HoBA-opt problem

```
Inputs  : The victim DNN $f$, the training dataset $X$,
          the number of perturbations $K$, and the
          perturbation magnitude $\xi$.
Outputs: The trained perturbations in $P$ and the
          trained selective neural network $g_\theta$.
1  Initialize each perturbation in $P$ by zeros.
2  Initialize $\theta$ by the Kaiming initialization [35].
3  do
4      Sample a batch of images $X_B$ from $X$.
5      Compute the loss on $X_B$ by (1).
6      Compute the gradients with respect to $P$.
7      Update $P$ by the projected gradient descent [36].
8      Repeat step-4 to sample a new batch of $X_B$.
9      Compute the loss on $X_B$ by (1).
10     Compute the gradients with respect to $\theta$.
11     Update $\theta$ by the ADAM optimizer [37].
12 while not converge;
13 return $P$ and $g_\theta$.
```

Algorithm 2: Conducting a boosting attack

```
Inputs : The trained perturbations in $P$, the trained
         selective neural network $g_\theta$ and a new target
         image $x_{\text{new}}$.
Output: The attacked image $x'_{\text{new}}$.
Compute: $\mathbb{P}_\theta(x_{\text{new}}) = g_\theta(x_{\text{new}})$.
Select: $j^* \leftarrow \text{argmax}_j\, \mathbb{P}_\theta(x_{\text{new}})_j$.
Perturb: $x'_{\text{new}} \leftarrow x_{\text{new}} + \delta_{j^*}$.
Clip the pixel values of $x'_{\text{new}}$ to the range of $[0, 255]$.
return $x'_{\text{new}}$.
```

[17], [22], [30], we use a surrogate DNN to train the learnable parameters of HoBA and HeBA in the black-box setting.

IV. Homogeneous Boosting Attack

In this section, we present the homogeneous boosting attack (HoBA) to address the FDUAP problem.

Denote by $L(x_i, \delta_j)$ a *loss function* that is used to train the perturbations in P, where x_i is the target image and $\delta_j \in P$ is the perturbation used to attack x_i. This function measures the similarity between the predictions of x_i made by the victim DNN f before and after δ_j is added to x_i [13]. A smaller value of $L(x_i, \delta_j)$ indicates f is more likely to change its prediction on x_i, thus the attack launched by adding δ_j to x_i is more likely to be successful [13], [30]. In the remainder of this paper, we refer to the loss function $L(x_i, \delta_j)$ as **attack function** to distinguish it from the final loss functions of the proposed boosting attack methods.

Since many effective attack functions have been proposed by different single-UAP methods in the literature [34], it is beyond the scope of this work to develop a new one. Our proposed boosting attack methods are generally compatible with the existing attack functions, and we adopt five attack functions proposed by the existing works [14], [21], [22], [24], [25] in our experiments.

To conduct HoBA based on the single attack function $L(x_i, \delta_j)$, we formulate the FDUAP problem as the following optimization problem named **HoBA-opt problem**:

$$\begin{aligned} \min_{P,\theta} \quad & \sum_{i=1}^{N}\sum_{j=1}^{K} \mathbb{P}_\theta(x_i)_j L(x_i, \delta_j), \\ \text{s.t.} \quad & \|\delta_j\|_\infty \leq \xi, \forall j \in \{1, \ldots, K\}, \end{aligned} \tag{1}$$

where $\mathbb{P}_\theta(x_i)_j$ represents the probability for a target image x_i to be a member of a cluster $S_j \in \mathcal{S}$, and the constraint $\|\delta_j\|_\infty \leq \xi$ follows the condition 3) in Definition 1.

We compute $\mathbb{P}_\theta(x_i)_j$ by a deep neural network g_θ named **selective neural network** that is parameterized by θ. Specifically, g_θ maps a target image x_i to a K-dimensional probability vector $\mathbb{P}_\theta(x_i)$, where the j-th entry is $\mathbb{P}_\theta(x_i)_j$.

For each target image $x_i \in X$, we assign x_i to the cluster S_j with the maximum probability $\mathbb{P}_\theta(x_i)_j$. This partitions the images in X into a set of K non-overlapping clusters, that is, $\mathcal{S} = \{S_1, \ldots, S_K\}$. Since all the images in a cluster S_j is attacked by the corresponding perturbation δ_j, solving the HoBA-opt problem finds the solutions P and θ that maximize the number of successfully attacked images in each of the K image clusters.

We solve the HoBA-opt problem by a training algorithm that alternately optimizes P and θ until the objective function in (1) becomes stable. Specifically, in each training iteration, we first update the K perturbations in P simultaneously by the projected gradient descent (PGD) [36] to handle the convex constraints $\|\delta_j\|_\infty \leq \xi$, and then we update θ by the ADAM optimizer [37], which is a classic method to train deep neural networks. The details of the proposed training algorithm are summarized in Algorithm 1.

Time complexity of Algorithm 1. Denoted by N the number of training images, by E the number of training epochs and by B the batch size. The total number of training iterations $I = NE/B$. The time cost of Algorithm 1 mainly consists of the time costs of the I training iterations, in each of which we use a batch of images X_B to optimize P and θ. Denoted by t_g the time cost of computing $\mathbb{P}_\theta(x_i)$ using g_θ, by t_l the time cost of computing the value of $L(x_i, \delta_j)$, and by d the number of parameters (i.e., pixels) of each perturbation in P, the major time cost of each training iteration is analyzed as follows.

In step 5, performing a forward pass to compute the loss $\sum_{j=1}^{K} \mathbb{P}_\theta(x_i)_j L(x_i, \delta_j)$ for each image in X_B takes $t_g + t_l K$ time. Thus, the time cost of step 5 to forward pass B images is $B(t_g + t_l K)$. In step 6, the time cost of computing the gradients with respect to P for X_B via backpropagation is roughly two times that of the forward passes in step 5, which is $2B(t_g + t_l K)$. In step 7, the time cost of updating P by the PGD is proportional to the number of parameters of the perturbations in P. Thus, the time cost of step 7 can be approximated by dK. Steps 9 and 10 have the same time costs as steps 5 and 6, respectively. In step 11, the time cost of updating θ by the ADAM optimizer is proportional to the number of parameters of g_θ. We denote this time cost as t_θ.

Summing up the time costs of the I training iterations, the time complexity of Algorithm 1 is $I(B(6t_g + 6t_l K) + dK + t_\theta) \sim O\big(I(B(t_g + t_l K) + dK + t_\theta)\big)$. Since d is a constant that is often dominated by t_l, t_θ is also a constant because the number of parameters of g_θ is fixed and we have $I = NE/B$, this time complexity can be simplified to $O\big(NE(t_g + t_l K)\big)$. Since N and E are often constants and t_g and t_l are not affected much by K, the time complexity of Algorithm 1 grows linearly with K. As shown by our experiments in Section VI, HoBA can significantly improve the attack effectiveness of the single-UAPs methods [14], [21], [22], [24], [25] by using only a small set of $K = 5$ perturbations, which indicates that the extra training time cost of HoBA is a cost-effective tradeoff.

Algorithm 1 trains P and g_θ in the white-box setting, which requires access to the architecture and parameters of the victim DNN f to compute gradients. This approach can be directly extended to the black-box setting by first training a surrogate DNN of f and then using the architecture and parameters of the surrogate DNN to train P and g_θ.

Given the trained P and g_θ, HoBA attacks a new target image x_{new} following the steps in Algorithm 2. We analyze the time complexity of Algorithm 2 as follows.

Time complexity of Algorithm 2. In step 1, the time cost of computing $\mathbb{P}_\theta(x_{\text{new}})$ using g_θ is t_g. In step 2, the time cost of selecting the most effective perturbation δ_{j^*} by performing an argmax operation for K elements is K. In step 3, the time cost of perturbing x_{new} by adding δ_{j^*} is d. In step 4, the time cost of clipping the pixel values of the attacked image is also d. To sum up, the time complexity of Algorithm 2 is $t_g + K + 2d \sim O\big(t_g + K + d\big)$. This indicates the fast attack speed of HoBA because the time cost t_g of feeding x_{new} into g_θ for a single forward pass is very small, which takes around 2.4 milliseconds on an NVIDIA RTX 3060 GPU in our experiments; K is small and usually not larger than 100; and the time cost d of adding the selected perturbation or clipping pixel values is also small, typically on the order of microseconds.

Meanwhile, HoBA also achieves good attack performance because of the high diversity of the perturbations in P. If each perturbation can successfully attack a substantially different set of target images, then HoBA will successfully attack the union of the images that are successfully attacked by each of the perturbations in P, as illustrated by the Venn diagram in Fig. 1(b.2).

The perturbations trained by HoBA are diversified because each of these perturbations is trained to focus on attacking a different cluster of images. To further enhance the attack effectiveness, we extend HoBA to an advanced version, which uses multiple attack functions together to train more diversified perturbations. This could offer higher attack effectiveness because the sets of target images successfully attacked by these perturbations are more distinct, which have less overlap and collectively cover a wider range of images, as illustrated by the Venn diagram in Fig. 1(b.3).

V. Heterogeneous Boosting Attack

In this section, we first introduce how to extend HoBA to the heterogeneous boosting attack (HeBA) by formulating a new HeBA-opt problem. Then, we describe how to solve the HeBA-opt problem and conduct HeBA attacks.

A. Formulating the HeBA-opt Problem

The key idea of HeBA is to use a number of H different attack functions to train a more diverse set of perturbations. We use each attack function to train a set of R perturbations, which results in a total number of $H \times R$ perturbations, denoted by $P = \{\delta_r^h \mid h \in \{1, \dots, H\}, r \in \{1, \dots R\}\}$. This approach significantly enhances the diversity of the perturbations in P, because the perturbations trained by different attack functions often tend to exhibit substantial variation [38].

Denote by $\mathcal{A} = \{L_h(x_i, \delta_r^h) \mid h \in \{1, \dots, H\}\}$ the set of attack functions, where each function $L_h(x_i, \delta_r^h)$ measures the similarity between the predictions of x_i made by the victim DNN f before and after δ_r^h is added to x_i. A smaller value of $L_h(x_i, \delta_r^h)$ indicates f is more likely to change its prediction on x_i, thus the attack launched by adding δ_r^h to x_i is more likely to be successful. In our experiments, the set $\mathcal{A}$ consists of the five different attack functions proposed by the existing single-UAP methods [14], [21], [22], [24], [25].

To conduct HeBA based on the set of attack functions $\mathcal{A}$, we extend the HoBA-opt problem to the following **HeBA-opt problem**:

$$\begin{aligned} \min_{P,\theta} \quad & \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{r=1}^{R} \mathbb{P}_\theta(x_i)_r^h L_h(x_i, \delta_r^h), \\ \text{s.t.} \quad & \left\|\delta_r^h\right\|_\infty \leq \xi, \forall h \in \{1, \dots, H\}, \forall r \in \{1, \dots, R\}, \end{aligned} \tag{2}$$

where $\mathbb{P}_\theta(x_i)_r^h$ presents the probability for a target image x_i to be a member of a cluster in $\mathcal{S}$, and the constraint $\left\|\delta_r^h\right\|_\infty \leq \xi$ follows the condition 3) of Definition 1.

To compute $\mathbb{P}_\theta(x_i)_r^h$, we extend the selective neural network $g_\theta(x_i)$ to produce a $(H \times R)$-dimensional probability vector and reshape it to a probability matrix $\mathbb{P}_\theta(x_i)$ with H rows and R columns. $\mathbb{P}_\theta(x_i)_r^h$ is the entry in the h-th row and r-th column of $\mathbb{P}_\theta(x_i)$.

Similar to HoBA, we can also use the probability matrix $\mathbb{P}_\theta(x_i)$ to partition X into a set of non-overlapping clusters, denoted by $\mathcal{S} = \{S_r^h \mid h \in \{1, \dots, H\}, r \in \{1, \dots, R\}\}$. This is done by assigning each image $x_i \in X$ to the cluster S_r^h with the maximum probability $\mathbb{P}_\theta(x_i)_r^h$ in $\mathbb{P}_\theta(x_i)$. Solving the HeBA-opt problem finds the solutions P and θ that maximize the number of successfully attacked images in each image cluster in $\mathcal{S}$.

Compared to HoBA, the perturbations of HeBA are not only trained to attack different clusters of images, but also trained with different attack functions. This enables the generation of more diversified perturbations, which can further improve attack effectiveness.

B. Solving the HeBA-opt Problem

The HeBA-opt problem in (2) can be solved by Algorithm 1, which was proposed in the *conference version* of our paper [1]. Denoted by t_g the time cost of computing $\mathbb{P}_\theta(x_i)$ and by t_l the worst-case time cost of computing the value of $L_h(x_i, \delta_r^h)$. Following the same time complexity analysis as in Section IV, the time complexity of solving the HeBA-opt problem by Algorithm 1 is $O\big(NE(t_g + t_l HR)\big)$.

In this *extended version* of our paper, we propose a more efficient new method to solve the HeBA-opt problem, which uses the *Gumbel softmax trick* [39], [40] to significantly reduce the time complexity of solving the HeBA-opt problem, thereby improving the training efficiency of HeBA. This new method consists of the following two steps.

First, we rewrite the term $\sum_{h=1}^{H}\sum_{r=1}^{R}\mathbb{P}_\theta(x_i)_r^h L_h(x_i,\delta_r^h)$ in (2) by its expectation form

$$\mathbb{E}_{z\sim g_\theta(x_i)}\Big[\sum_{h=1}^{H}(z_1^h+\ldots+z_R^h)L_h(x_i,\delta_1^h z_1^h+\ldots+\delta_R^h z_R^h)\Big], \tag{3}$$

where z is a $(H\times R)$-dimensional one-hot vector that models a categorical variable following the distribution of $\mathbb{P}_\theta(x_i)$. Since $\mathbb{P}_\theta(x_i)=g_\theta(x_i)$, we write $z\sim g_\theta(x_i)$. We reshape z as a discrete matrix with H rows and R columns in the same way as we reshaped $\mathbb{P}_\theta(x_i)$ into a matrix. Denote by z_r^h the entry in the h-th row and r-th column of the matrix z, $z_r^h=1$ means $\delta_r^h\in P$ is selected by g_θ to attack x_i.

Second, we use the Gumbel softmax trick to approximate the discrete matrix z by a smooth matrix s. This approximates each entry z_r^h in z by

$$s_r^h=\frac{\exp\big(\big(\log\mathbb{P}_\theta(x_i)_r^h+\epsilon_r^h\big)/\tau\big)}{\sum_{a=1}^{H}\sum_{b=1}^{R}\exp\big(\big(\log\mathbb{P}_\theta(x_i)_b^a+\epsilon_b^a\big)/\tau\big)}, \tag{4}$$

where ϵ is a matrix of size $H\times R$, with each entry ϵ_r^h being an independent random variable following the $Gumbel(0,1)$ distribution [39], and $\tau>0$ is the temperature parameter that controls the smoothness of s_r^h. When τ gets closer to zero, the entries in s get closer to 0 or 1.

Through the above two steps, we convert the original HeBA-opt problem into

$$\begin{aligned}\min_{P,\theta}\quad&\sum_{i=1}^{N}\mathbb{E}_{\epsilon\sim Gumbel(0,1)}\Big[\sum_{h=1}^{H}(s_1^h+\ldots+s_R^h)\\&L_h(x_i,\delta_1^h s_1^h+\ldots+\delta_R^h s_R^h)\Big],\\\text{s.t.}\quad&\left\|\delta_r^h\right\|_\infty\le\xi,\forall h\in\{1,\ldots,H\},\forall r\in\{1,\ldots,R\},\end{aligned}\tag{5}$$

which is solved by the gradient-based method in Algorithm 3.

The key idea of Algorithm 3 is to alternatively update P and θ until the objective function in (5) becomes stable. In each training iteration, we first update all the $H\times R$ perturbations in P by the PGD and then update θ by the ADAM optimizer. Following the routine of the Gumbel softmax trick [39], for each update, the value of the objective function is computed on a batch of images X_B and a matrix ϵ that is sampled from $Gumbel(0,1)$. Then, we compute the gradients by standard backpropagation. The temperature τ is gradually reduced to avoid introducing a large variance in the gradients computed early in the training and to make the smooth matrix s closer to discrete as the training proceeds [39], [40].

Time complexity of Algorithm 3. The time cost of Algorithm 3 mainly consists of the time costs of the $I=NE/B$ training iterations, in each of which we use a batch of images X_B to optimize P and θ. We analyze the major time cost of each training iteration as follows.

In step 7, the time cost of performing a forward pass to compute $\mathbb{E}_{\epsilon\sim Gumbel(0,1)}[\cdot]$ for each image in X_B is t_g+HR+t_lH because it takes t_g time to compute $\mathbb{P}_\theta(x_i)$ using g_θ, HR time to compute the entries in s by (4) and t_lH time to compute $L_h(x_i,\delta_1^h s_1^h+\ldots+\delta_R^h s_R^h)$ for H times. Thus, the time cost of step 7 to forward pass B images is $B(t_g+HR+t_lH)$. In step 8, the time cost of computing the gradients of P is roughly two times that of the forward passes in step 7, which is $2B(t_g+HR+t_lH)$. In step 9, the time cost of updating P by the PGD is proportional to the number of parameters of the perturbations in P, which can be approximated by dHR. Steps 11 and 12 have the same time cost as steps 7 and 8, respectively. In step 13, the time cost of updating θ by the ADAM optimizer is proportional to the number of parameters of g_θ and we denote this time cost as t_θ.

```
Algorithm 3: Solving the HeBA-opt problem
Inputs  : The victim DNN $f$, the training dataset $X$,
          the set of $H$ attack functions $\mathcal{A}$, the number
          of perturbations $R$ for each attack function,
          and the perturbation magnitude $\xi$.
Outputs: The trained perturbations in $P$ and the
          trained selective neural network $g_\theta$.
1  Initialize each perturbation in $P$ by zeros.
2  Initialize $\theta$ by the Kaiming initialization [35].
3  Initialize the temperature parameter $\tau = 1$.
4  Set the number of training iterations $t = 1$.
5  do
6      Sample a batch of images $X_B$ from $X$ and sample
         $\epsilon$ from the distribution $Gumbel(0,1)$.
7      Compute the loss in (5) using $X_B$ and $\epsilon$.
8      Compute the gradients with respect to $P$.
9      Update $P$ by the projected gradient descent [36].
10     Repeat step-6 to sample new $X_B$ and $\epsilon$.
11     Compute the loss in (5) using $X_B$ and $\epsilon$.
12     Compute the gradients with respect to $\theta$.
13     Update $\theta$ by the ADAM optimizer [37].
14     If $t \bmod 2000 = 0$ then update $\tau$ by
         $\tau \leftarrow \max\big(0.01, \exp(-1e-4\cdot t)\big)$
15     Update $t \leftarrow t+1$.
16 while not converge;
17 return $P$ and $g_\theta$.
```

Summing up the time costs of the I training iterations, the time complexity of Algorithm 3 is $I(B(6t_g+6t_lH+6HR)+dHR+t_\theta)\sim O\big(I(B(t_g+t_lH+HR)+dHR+t_\theta)\big)$. Since d is a constant that is often dominated by t_l, R is typically small and often not exceeding 20, t_θ is also a constant because the number of parameters of g_θ is fixed and we have $I=NE/B$, this time complexity can be simplified to $O\big(NE(t_g+t_lH)\big)$. Since N and E are often constants and t_g and t_l are not affected much by H, the time complexity of Algorithm 3 grows linearly with respect to H. This indicates the higher efficiency of Algorithm 3 in solving the HeBA-opt problem compared to Algorithm 1. We compare the efficiency of Algorithm 1 and Algorithm 3 in practice in Appendix A.

Algorithm 3 trains P and g_θ in the white-box setting. We extend it to the black-box setting by using a surrogate DNN in the same way as we extended Algorithm 1. After training, the process of conducting a boosting attack by HeBA is the same as that of HoBA described in Algorithm 2, and thus we omit it here to avoid redundancy.

VI. Experiments

In this section, we systematically evaluate the performance of the proposed boosting attack framework. In particular, we wish to answer the following research questions. **Q1:** How effective is our framework in the white-box setting? **Q2:** How effective is our framework in the black-box setting? **Q3:** How effective is our framework against typical defenses? **Q4:** How fast is the boosting attack? **Q5:** How diversified are the UAPs generated by our framework? **Q6:** Does the diversity of UAPs relate to the attack effectiveness? **Q7:** How does our framework perform when using different numbers of UAPs? **Q8:** How does our framework perform compared to the single-UAP methods that use more training images? **Q9:** What is the distribution of images attacked by different UAPs? **Q10:** How does our framework perform when using different model architectures for the selective neural network g_θ? **Q11:** What is the training time of different attack methods? **Q12:** How effective is our framework in the objection detection task? **Q13:** How effective is our framework in the semantic segmentation task? **Q14:** What is the potential limitation of our framework? Due to the page limit, we answer Q9-Q14 in Appendices B-G of the *supplementary material*.

A. Experimental Settings

Baselines. We compare the proposed boosting attack framework with five representative single-UAP methods, including UAT [21], DF-UAP [25] written in short as DF, Cosine-UAP [24] written in short as COS, NAG [22], and TRM-UAP [14] written in short as TRM. We also compare with two image-dependent generator-based methods, including GAP [29] and TDA [28].

Our methods. For HoBA, we apply it on each of the above single-UAP methods by adopting the same attack function $L(x_i, \delta_j)$ as the single-UAP method when solving the HoBA-opt problem in (1). To be specific, we implement HoBA in the following two different types.

- **HoBA type 1.** We first run each single-UAP method to generate a set of K UAPs. For UAT, DF, COS and TRM, we independently run each method for K times. For NAG, since it trains a generator to produce UAPs, we independently sample K times using the trained generator. After obtaining the K UAPs, we regard them as a constant set of perturbations P, and solve the HoBA-opt problem in (1) to train the selective neural network g_θ. That is, we do not execute steps 4-7 when calling Algorithm 1. Thus, this type can be viewed as an *ablated version* of HoBA since it does not train P and g_θ together.
- **HoBA type 2.** This is the *complete version* of HoBA as described in Algorithm 1, which solves the HoBA-opt problem in (1) to train P and g_θ together.

For HeBA, we adopt the attack functions of the above five single-UAP methods as the set of different attack functions $\mathcal{A} = \{L_h(x_i, \delta_r^h) \mid h \in \{1, \dots, H\}\}$ used in (5), where $H = 5$ since $\mathcal{A}$ contains five attack functions. We also implement HeBA in the following two different types.

- **HeBA type 1.** We first run each single-UAP method to generate a set of R UAPs in the same way as in HoBA type 1. After obtaining the $H \times R$ UAPs, we regard them as a constant set of perturbations P and solve the HeBA-opt problem in (5) to train the selective neural network g_θ. That is, we do not execute steps 6-9 when calling Algorithm 3. We view this type as an *ablated version* of HeBA since it does not train P and g_θ together.
- **HeBA type 2.** This is the *complete version* of HeBA as described in Algorithm 3, which solves the HeBA-opt problem in (5) to train P and g_θ together.

We denote by UAT-HoBA1, DF-HoBA1, COS-HoBA1, NAG-HoBA1 and TRM-HoBA1 the boosting attack methods when applying HoBA type 1 on each of the single-UAP methods; and by UAT-HoBA2, DF-HoBA2, COS-HoBA2, NAG-HoBA2 and TRM-HoBA2 the boosting attack methods when applying HoBA type 2. When the context is clear, we omit the names of the single-UAP methods and directly use HoBA1 and HoBA2 to refer to the corresponding boosting attack methods of a single-UAP method. We denote by HeBA1 and HeBA2 the boosting attack methods when applying HeBA in type 1 and type 2, respectively.

Attack model. We use the term "attack model" to denote the learnable parameters that need to be trained in each attack method. For our boosting attack methods, the attack model consists of the perturbations in P and the selective neural network g_θ. For each single-UAP method, its attack model is the single UAP that needs to be learned. For GAP and TDA, their attack models are the DNN-based generator that produces a perturbation for each target image.

Datasets. We use ImageNet [41] and COCO [42] to construct the following datasets: 1) training dataset D_1 consists of 10,000 images uniformly sampled from the original training dataset of ImageNet; 2) training dataset D_2 consists of 10,000 images uniformly sampled from the original training dataset of COCO; and 3) testing dataset D_3 consists of 50,000 images from the original validation dataset of ImageNet.

Usage of datasets. In the white-box setting, a victim DNN is trained on D_1, each attack model is trained on D_1, and the performance of the attack model is evaluated on D_3 when attacking the victim DNN. In the black-box setting, the victim DNN and the surrogate DNN are independently trained on D_1, each attack model is trained on D_2 to attack the surrogate DNN, and the performance of the attack model is evaluated on D_3 when attacking the victim DNN. Using D_2 instead of D_1 to train attack models enables us to assess the performance of the attack models when they are trained on a dataset that is different from the original training dataset of the victim DNN.

Victim DNNs. We use eight victim DNNs to evaluate the performance of different attack methods, including ResNet-50 [43], VGG-16 [44], Inception-V3 [45], SqueezeNet [46], ViT-B [47] and LeViT-128 [48], Swin-T [49] and MaxViT-T [50]. All the victim DNNs are not modified during the training and evaluation of the attack models.

Selective neural network. We adopt a SqueezeNet [46] as the architecture of the selective neural network g_θ. SqueezeNet is a lightweight convolutional neural network that consists of 18 layers. Specifically, it consists of one initial convolutional layer with a kernel size of 7×7 and a stride of 2, three max-pooling layers with a kernel size of 3×3 and a stride of 2, and seven fire modules [46], each composed of a squeeze

TABLE II
THE FR (%) WHEN $\xi \in \{2, 6, 10\}$. BOLD NUMBER MARKS THE HIGHEST FR IN EACH ROW. UNDERLINED NUMBER SHOWS THE RUNNER UP.

	ξ	GAP	TDA	(UAT, HoBA1, HoBA2)	(DF, HoBA1, HoBA2)	(COS, HoBA1, HoBA2)	(NAG, HoBA1, HoBA2)	(TRM, HoBA1, HoBA2)	(HeBA1, HeBA2)
ResNet-50	2	25.5	27.8	(19.0, 21.1, 24.7)	(21.6, 24.2, $\underline{29.0}$)	(21.2, 24.6, 28.3)	(18.5, 24.5, 27.6)	(20.6, 22.8, 25.3)	(26.4, **31.3**)
	6	84.2	84.4	(73.6, 77.6, $\underline{87.6}$)	(72.7, 76.6, 87.0)	(66.6, 71.8, 80.5)	(68.5, 75.7, 79.2)	(71.9, 75.2, 87.2)	(80.5. **89.4**)
	10	96.5	98.4	(94.1, 96.7, 98.1)	(94.6, 96.8, $\underline{99.0}$)	(92.3, 94.6, 97.7)	(91.1, 96.3, 97.9)	(93.2, 95.3, 98.1)	(97.2, **99.3**)
VGG-16	2	41.0	40.7	(36.2, 38.5, $\underline{44.0}$)	(35.3, 37.9, 42.7)	(32.6, 35.3, 42.6)	(35.1, 39.1, 42.9)	(34.6, 37.0, 42.4)	(41.6, **46.4**)
	6	89.1	87.5	(83.8, 87.5, 93.0)	(85.6, 89.4, 94.5)	(87.1, 90.4, 95.3)	(82.5, 91.5, $\underline{95.8}$)	(86.6, 89.5, 94.1)	(92.5, **96.3**)
	10	98.7	99.0	(95.3, 96.2, 98.5)	(94.1, 96.7, $\underline{99.4}$)	(95.4, 97.3, 99.2)	(92.2, 97.2, 99.1)	(95.0, 96.5, 99.0)	(98.0, **99.5**)
Inception-V3	2	34.2	35.4	(23.6, 26.9, 34.7)	(26.4, 29.5, $\underline{38.6}$)	(26.4, 29.9, 37.0)	(24.0, 30.2, 37.0)	(24.3, 28.3, 36.1)	(32.7, **40.0**)
	6	81.7	84.9	(73.3, 78.6, $\underline{85.4}$)	(72.2, 76.7, 85.1)	(70.0, 74.4, 83.8)	(70.3, 78.8, 84.3)	(68.2, 74.6, 81.8)	(79.7, **87.2**)
	10	95.4	97.8	(94.4, 97.5, $\underline{99.5}$)	(93.7, 95.5, 99.0)	(92.8, 95.4, 98.3)	(90.3, 96.3, 99.0)	(88.5, 93.0, 96.2)	(98.4, **99.6**)
SqueezeNet	2	33.3	36.5	(26.0, 29.9, $\underline{36.9}$)	(25.3, 29.0, 35.3)	(27.5, 30.0, 35.4)	(24.2, 33.7, 34.3)	(23.6, 28.7, 33.5)	(34.9, **37.5**)
	6	87.5	89.0	(80.0, 83.7, 90.6)	(79.3, 82.2, $\underline{91.2}$)	(77.5, 81.8, 89.2)	(76.3, 83.2, 88.3)	(74.9, 83.2, 88.4)	(84.4, **93.3**)
	10	98.3	98.6	(94.9, 96.3, 98.8)	(94.3, 96.1, $\underline{99.5}$)	(93.8, 95.0, 99.4)	(91.0, 95.3, 99.3)	(92.4, 94.5, 97.9)	(97.2, **99.7**)
ViT-B	2	35.5	36.1	(25.3, 28.6, 35.1)	(29.5, 32.4, $\underline{40.3}$)	(25.6, 31.8, 39.2)	(27.9, 30.4, 37.9)	(24.3, 27.7, 33.8)	(32.9, **43.4**)
	6	83.7	84.4	(76.5, 80.4, $\underline{88.9}$)	(75.6, 79.3, 88.3)	(73.4, 78.1, 86.2)	(73.6, 80.3, 86.5)	(73.5, 76.0, 83.4)	(81.3, **89.8**)
	10	96.7	97.9	(95.2, 98.6, **99.6**)	(95.1, 98.3, $\underline{99.5}$)	(94.9, 97.8, 99.1)	(94.3, 98.7, 99.2)	(94.5, 96.5, 98.7)	(98.7, **99.6**)
LeViT-128	2	34.9	35.8	(24.6, 27.7, 36.2)	(27.4, 31.2, $\underline{40.2}$)	(26.1, 30.5, 39.0)	(25.7, 31.4, 37.7)	(24.8, 29.6, 36.5)	(33.3, **41.9**)
	6	81.3	83.6	(74.2, 78.1, $\underline{87.6}$)	(73.5, 78.5, 85.9)	(73.8, 76.6, 84.3)	(71.2, 79.6, 84.4)	(72.5, 79.2, 85.1)	(81.1, **89.0**)
	10	96.3	98.2	(94.6, 98.2, $\underline{99.1}$)	(94.2, 98.0, **99.6**)	(93.9, 97.6, 98.9)	(93.8, 98.0, 99.0)	(92.4, 95.3, 98.4)	(98.7, **99.6**)
Swin-T	2	34.4	38.7	(26.5, 30.3, 39.8)	(30.7, 36.4, $\underline{45.5}$)	(27.6, 30.6, 39.4)	(29.7, 31.3, 38.7)	(27.2, 31.6, 39.5)	(37.5, **47.5**)
	6	85.4	87.0	(75.3, 81.4, 89.5)	(77.4, 81.6, $\underline{90.7}$)	(74.3, 80.6, 88.5)	(71.7, 76.0, 85.9)	(75.4, 79.7, 88.2)	(82.5, **91.3**)
	10	97.2	97.9	(94.7, 98.1, 99.3)	(95.6, 98.5, $\underline{99.5}$)	(94.3, 97.9, 99.3)	(94.4, 98.2, 99.1)	(95.4, 97.7, 99.0)	(98.2, **99.7**)
MaxViT-T	2	36.3	39.2	(27.5, 31.2, 37.7)	(31.2, 35.4, $\underline{43.2}$)	(26.5, 33.0, 41.8)	(28.5, 31.7, 39.6)	(26.5, 30.2, 36.5)	(36.8, **46.0**)
	6	87.1	88.6	(78.3, 83.7, $\underline{89.2}$)	(79.8, 82.7, 87.2)	(72.4, 77.7, 86.9)	(76.3, 82.0, 88.9)	(72.5, 77.0, 87.7)	(85.0, **90.8**)
	10	98.2	97.3	(95.2, 97.2, 99.5)	(94.7, 98.3, $\underline{99.6}$)	(94.2, 96.5, 98.6)	(93.5, 96.7, 99.3)	(94.6, 96.7, 98.9)	(98.8, **99.7**)

layer followed by an expand layer. It has been shown to perform well in many different vision tasks [51], [52] and we empirically find good performance of our boosting attack methods in the experiments when employing SqueezeNet as g_θ. For HoBA methods, we use K softmax-activated output neurons in the last layer of SqueezeNet; and for HeBA methods, we use $H \times R$ softmax-activated output neurons in the last layer of SqueezeNet. We also investigate the impact of different model architectures of g_θ on the performance of our boosting attack methods in Appendix C.

Evaluation metrics. Following the previous works [6], [21], [22], we adopt the *fooling ratio (FR)* on the testing dataset D_3 to evaluate attack effectiveness. It is computed as the proportion of images in D_3 that are successfully attacked. A larger FR indicates better attack effectiveness. We report FR in percentage by default.

We evaluate the *average attack time (AAT)* of each attack method by measuring the average time cost of generating the perturbed image of a target image in D_3. Since the attack model of each attack method is trained offline before conducting attacks, AAT does not include the time to train the attack model. The training time of attack methods is discussed in Appendix D. A smaller AAT implies a faster attack speed and we report AAT in milliseconds by default.

Implementation details. For the baseline methods, we use the default hyperparameters used in their implementations [14], [21], [22], [24], [25], [28], [29]. For our boosting attack methods, if not otherwise specified, we use $K = 5$ for HoBA methods and $H = 5$, $R = 1$ for HeBA methods. We also study the effect of different values of K and R in Section VI-H. By default, we set the number of training epochs to 100 and set the learning rates of the PGD and the ADAM optimizer to 10^{-3}.

For HoBA methods, we use a batch size of $\lfloor \frac{64}{K} \rfloor$ due to the limit of GPU memory. This is because, when computing the loss value of (1) for each training image, we need to compute the value of the attack function $L(x_i, \delta_j)$ for K times by adding each of the K perturbations to that image and then feeding the K perturbed images to a victim DNN. Therefore, the number of perturbed images fed to the victim DNN in one training iteration is K times of the batch size. Since our GPU memory can accommodate at most 64 perturbed images fed to the victim DNN at the same time, $\lfloor \frac{64}{K} \rfloor$ is the maximum batch size that can be accommodated by our GPU. Due to the same reason, for HeBA methods, we can derive from (5) that the number of images fed to a victim DNN in one iteration is H times of the batch size. Thus, we set the batch size to $\lfloor \frac{64}{H} \rfloor$. The implementations are realized using Pytorch version 1.11.0 with CUDA version 11.3. All experiments are conducted on a server with an NVIDIA 3060 GPU, 32GB main memory, and an Intel(R) Core(TM) i9-10900F CPU @ 2.80GHz.

B. Attack Effectiveness in the White-box Setting (Q1)

In this subsection, we answer Q1 by comparing the attack effectiveness of our boosting attack methods with the baseline methods in the white-box setting. Table II reports the FR of all the attack methods when $\xi \in \{2, 6, 10\}$. From the results, we have the following observations.

We can see that when HoBA1 and HoBA2 are applied, the FR of the single-UAP methods is significantly improved. In addition, HeBA1 and HeBA2 also achieve much higher FR than that of the single-UAP methods. Moreover, the FR of UAT-HoBA2, DF-HoBA2 and HeBA2 is almost always higher than that of the image-dependent generator-based methods GAP and TDA, and the highest FR is consistently achieved by HeBA2 as shown by the bold numbers in each row of Table II. These results demonstrate the great performance of the proposed boosting attack framework in improving the attack effectiveness of the single-UAP methods.

Comparing HoBA2 and HeBA2 with their type 1 counterparts, we can see that HoBA2 consistently achieves a higher FR than that of HoBA1, and HeBA2 consistently outperforms HeBA1. This is because the set of UAPs used by HoBA1 and

TABLE III

THE TRANS$_{f_i}$ (%) WHEN $\xi \in \{2, 6, 10\}$. BOLD NUMBER MARKS THE HIGHEST TRANS$_{f_i}$ IN EACH ROW. UNDERLINED NUMBER SHOWS THE RUNNER UP.

	f_i	GAP	TDA	(UAT, HoBA1, HoBA2)	(DF, HoBA1, HoBA2)	(COS, HoBA1, HoBA2)	(NAG, HoBA1, HoBA2)	(TRM, HoBA1, HoBA2)	(HeBA1, HeBA2)
$\xi = 2$	ResNet-50	10.8	12.9	(7.8, 9.4, 12.4)	(7.7, 9.7, 12.8)	(8.0, 10.6, $\underline{14.0}$)	(6.9, 9.6, 12.4)	(7.7, 9.9, 12.2)	(11.7, **16.0**)
	VGG-16	13.9	18.6	(12.7, 16.5, $\underline{20.4}$)	(13.0, 14.6, 19.9)	(12.9, 15.5, 19.5)	(12.7, 15.6, 18.7)	(13.1, 15.8, 19.6)	(18.3, **22.3**)
	Inception-V3	17.9	19.2	(13.7, 15.9, $\underline{20.3}$)	(15.3, 16.8, 19.6)	(14.0, 16.6, 19.7)	(14.3, 16.3, 19.8)	(14.4, 16.6, 19.9)	(18.6, **23.1**)
	SqueezeNet	15.1	19.4	(12.2, 14.6, 18.9)	(13.7, 14.6, 19.6)	(13.6, 16.0, $\underline{20.3}$)	(12.6, 15.2, 18.9)	(13.3, 15.4, 19.4)	(15.8, **22.7**)
	ViT-B	7.0	8.0	(5.9, 6.6, $\underline{10.4}$)	(6.3, 7.6, $\underline{10.4}$)	(5.5, 7.7, 10.2)	(5.2, 7.4, 9.9)	(6.1, 7.6, 10.3)	(9.0, **13.4**)
	LeViT-128	7.4	9.2	(6.8, 8.0, 9.8)	(6.4, 8.6, 10.2)	(5.7, 7.6, $\underline{10.9}$)	(5.5, 7.8, 10.4)	(6.3, 8.0, 10.7)	(10.4, **12.9**)
	Swin-T	6.7	7.5	(6.4, 7.5, 9.0)	(7.2, 8.0, $\underline{9.7}$)	(5.8, 7.0, 8.8)	(6.2, 6.9, 8.4)	(6.6, 7.6, 9.3)	(8.9, **11.0**)
	MaxViT-T	7.2	8.7	(5.9, 7.2, 10.8)	(6.8, 8.0, 11.2)	(6.3, 8.3, $\underline{11.6}$)	(6.0, 7.6, 10.9)	(6.6, 8.0, 11.0)	(10.2, **12.6**)
$\xi = 6$	ResNet-50	50.8	57.0	(44.2, 48.6, $\underline{58.4}$)	(45.2, 48.0, 55.8)	(41.5, 45.3, 55.3)	(43.0, 48.5, 52.3)	(44.1, 48.8, 57.0)	(53.5, **60.0**)
	VGG-16	35.9	39.7	(34.4, 37.1, $\underline{42.4}$)	(32.5, 33.8, 39.2)	(31.6, 35.5, 39.6)	(31.0, 34.0, 41.7)	(34.0, 36.3, 41.1)	(39.2, **43.8**)
	Inception-V3	54.7	59.0	(49.7, 52.6, $\underline{60.2}$)	(47.1, 49.5, 57.4)	(45.2, 48.6, 54.3)	(45.3, 50.8, 57.2)	(48.2, 51.1, 58.0)	(55.9, **62.3**)
	SqueezeNet	45.6	51.8	(43.1, 46.6, $\underline{52.5}$)	(38.5, 39.0, 49.0)	(37.3, 41.2, 49.2)	(41.0, 42.4, 50.2)	(41.2, 42.8, 50.5)	(46.9, **53.4**)
	ViT-B	28.9	29.9	(24.9, 27.0, 32.7)	(26.0, 30.7, $\underline{33.2}$)	(23.2, 24.6, 28.8)	(24.8, 26.7, 31.2)	(26.0, 28.2, 31.3)	(29.5, **35.6**)
	LeViT-128	30.3	32.7	(26.5, 30.1, 33.8)	(27.2, 31.9, $\underline{36.0}$)	(24.9, 27.2, 32.4)	(28.3, 29.2, 35.2)	(28.1, 29.8, 35.8)	(33.1, **37.4**)
	Swin-T	29.4	31.5	(26.6, 29.8, 34.0)	(28.1, 30.4, $\underline{35.6}$)	(25.6, 27.5, 33.0)	(26.2, 28.6, 32.3)	(28.2, 30.1, 34.5)	(31.7, **37.8**)
	MaxViT-T	28.2	30.5	(23.5, 26.9, 31.0)	(26.2, 29.3, $\underline{34.8}$)	(21.5, 23.6, 29.7)	(24.9, 28.0, 32.0)	(24.9, 28.7, 32.6)	(30.2, **36.9**)
$\xi = 10$	ResNet-50	60.8	66.7	(61.3, 63.9, $\underline{67.8}$)	(59.3, 62.2, 66.8)	(57.7, 60.0, 66.8)	(60.6, 63.2, 66.0)	(61.8, 63.4, 65.8)	(65.3, **72.2**)
	VGG-16	43.0	49.3	(42.5, 44.6, 52.4)	(41.7, 43.0, $\underline{53.3}$)	(41.5, 45.3, 49.7)	(41.0, 44.8, 47.4)	(42.9, 45.1, 51.2)	(46.6, **55.8**)
	Inception-V3	66.2	69.7	(64.4, 67.6, $\underline{72.6}$)	(61.8, 64.0, 68.8)	(61.6, 65.5, 71.9)	(61.0, 66.9, 70.8)	(64.7, 67.4, 71.2)	(69.4, **74.8**)
	SqueezeNet	54.0	59.3	(48.0, 50.4, 59.5)	(52.0, 54.0, $\underline{62.7}$)	(48.2, 51.0, 57.6)	(50.7, 55.9, 60.4)	(50.0, 54.2, 60.2)	(57.5, **64.5**)
	ViT-B	32.7	38.2	(32.9, 36.3, 40.6)	(30.4, 36.7, 41.6)	(29.5, 36.9, $\underline{43.7}$)	(32.6, 35.6, 41.8)	(31.7, 37.5, 41.7)	(39.5, **45.8**)
	LeViT-128	39.3	42.0	(37.5, 40.8, 45.5)	(41.1, 42.8, $\underline{45.8}$)	(36.3, 40.3, 44.8)	(36.3, 39.9, 42.5)	(37.9, 39.1, 44.0)	(44.4, **47.0**)
	Swin-T	30.4	35.2	(29.6, 32.2, 37.9)	(28.5, 33.0, 37.4)	(27.3, 31.6, 34.0)	(30.2, 34.3, $\underline{38.0}$)	(30.4, 35.1, 37.3)	(35.8, **39.6**)
	MaxViT-T	36.5	41.4	(34.6, 38.9, 43.8)	(36.2, 39.8, $\underline{44.6}$)	(34.2, 37.0, 41.2)	(35.1, 37.9, 42.8)	(36.8, 39.6, 44.1)	(41.5, **46.7**)

HeBA1 is separately trained from the selective neural network g_θ, with each UAP trained by an independent run of a single-UAP method on the entire training dataset D_1. The UAPs generated in this way usually do not have very high diversity, thus reducing the FR of HoBA1 and HeBA1. For HoBA2 and HeBA2, the set of UAPs is trained together with g_θ. These UAPs tend to have high diversity as each of the UAPs is trained to attack the images in a separate cluster of images, thus leading to much higher FR of HoBA2 and HeBA2.

Comparing the performance of HoBA and HeBA methods, we can see that HeBA1 and HeBA2 often achieve higher FR than that of HoBA1 and HoBA2, respectively. This demonstrates the superiority of HeBA methods, which is achieved via further enhancing the diversity of UAPs by training them using multiple different types of attack functions. Compared with the UAPs generated by HoBA methods, the UAPs generated by HeBA methods are able to successfully attack more diversified sets of images, thereby attaining higher attack effectiveness. We will analyze the diversity of the UAPs trained by our boosting attack methods in Section VI-F.

C. Attack Effectiveness in the Black-box Setting (Q2)

In this subsection, we answer Q2 by evaluating the attack effectiveness of different attack methods in the black-box setting, where an attacker has no access to the architecture or the parameters of the victim DNN. Following the routine of transfer adversarial attacks [16], [17], [22], [30], we use a surrogate DNN to train the attack model of each attack method in this setting. Since the attacker lacks knowledge of the victim DNN's architecture, we use the architecture of each victim DNN as the surrogate DNN's architecture and report the attack effectiveness of all cases.

Denote by FR$_{i,j}$ ($i \neq j$) the transfer attack effectiveness when transferring from a surrogate DNN f_i to a victim DNN f_j. For each attack method, we compute FR$_{i,j}$ in the following steps: 1) train f_i on D_1; 2) train the attack model to attack f_i on D_2; and 3) use the trained attack model to attack f_j on D_3 and calculate the corresponding FR as FR$_{i,j}$.

Denote by $Q = \{f_1, \ldots, f_8\}$ the eight victim DNNs used in our experiments. For each attack method, we measure its *average transfer attack effectiveness* when transferring from a surrogate DNN $f_i \in Q$ to the other victim DNNs $f_j \in Q \setminus f_i$ by

$$\text{Trans}_{f_i} = \frac{1}{7} \sum_{f_j \in Q \setminus f_i} \text{FR}_{i,j}, \quad (6)$$

which is the average of FR$_{i,j}$ when transferring from the surrogate DNN f_i to each of the other victim DNNs. A higher Trans$_{f_i}$ means better transfer attack effectiveness.

Table III reports the Trans$_{f_i}$ of different attack methods when $\xi \in \{2, 6, 10\}$. We can see that all the boosting attack methods consistently outperform the single-UAP methods, and the highest Trans$_{f_i}$ is consistently achieved by HeBA2. These results demonstrate the good performance of the boosting attack framework in enhancing the transfer attack effectiveness of the single-UAP methods.

We explain the superior transferability of our boosting attack methods as follows. *First,* due to the good transferability of UAP [14], [15], [17], each of the diversified UAPs transfers well from attacking the surrogate DNN to the other victim DNNs. *Second,* since these UAPs are diversified, they tend to successfully transfer-attack complementary sets of images. *Third,* g_θ also transfers well in selecting the most effective UAP to successfully attack different victim DNNs. Therefore, the boosting attack methods achieve superior transferability by successfully attacking the union of the images successfully transfer-attacked by each of the diversified UAPs.

In addition, we observe a similar phenomenon to that in Table II, where HoBA2 and HeBA2 consistently outperform HoBA1 and HeBA1, respectively. Furthermore, in most cases, HeBA1 and HeBA2 obtain higher Trans$_{f_i}$ than that of HoBA1 and HoBA2, respectively. This suggests that the high diversity of UAPs is an important factor contributing to the high transfer attack effectiveness. We will analyze the diversity of the UAPs trained by our boosting attack methods in Section VI-F.

TABLE IV
THE FR (%) IN THE PRESENCE OF DIFFERENT DEFENSES WHEN $\xi = 6$. BOLD NUMBER MARKS THE HIGHEST FR IN EACH ROW. UNDERLINED NUMBER SHOWS THE RUNNER UP.

	Defense	GAP	TDA	(UAT, HoBA1, HoBA2)	(DF, HoBA1, HoBA2)	(COS, HoBA1, HoBA2)	(NAG, HoBA1, HoBA2)	(TRM, HoBA1, HoBA2)	(HeBA1, HeBA2)
ResNet-50	No defense	84.2	84.4	(73.6, 77.6, 87.6)	(72.7, 76.6, 87.0)	(66.6, 71.8, 80.5)	(68.5, 75.7, 79.2)	(71.9, 75.2, 87.2)	(80.5, **89.4**)
	JPEG	46.5	44.7	(36.3, 42.8, 47.7)	(34.2, 40.5, 46.8)	(29.6, 34.4, 40.9)	(31.6, 37.6, 42.3)	(28.7, 32.2, 39.6)	(45.2, **49.7**)
	Crop and rescale	28.8	25.3	(22.2, 26.7, 30.4)	(22.4, 27.4, 31.7)	(16.3, 22.9, 26.0)	(18.4, 23.8, 28.5)	(19.2, 22.6, 27.0)	(27.4, **32.8**)
	TV minimization	35.9	37.3	(29.2, 32.1, 38.6)	(28.5, 32.2, 39.2)	(23.3, 26.4, 30.1)	(25.2, 28.8, 33.2)	(26.2, 30.4, 34.7)	(36.0, **39.5**)
	Bit depth reduction	45.7	45.3	(35.2, 40.4, 46.7)	(36.8, 42.3, 47.5)	(27.6, 33.6, 38.2)	(30.7, 37.0, 42.4)	(31.5, 33.0, 36.9)	(44.6, **49.0**)
	Adversarial training	40.3	38.2	(32.2, 36.5, 41.9)	(34.5, 37.2, 43.5)	(31.1, 34.7, 38.0)	(32.6, 37.7, 42.0)	(33.1, 37.6, 42.7)	(42.4, **46.4**)
VGG-16	No defense	89.1	87.5	(83.8, 87.5, 93.0)	(85.6, 89.4, 94.5)	(87.1, 90.4, 95.3)	(82.5, 91.5, 95.8)	(86.6, 89.5, 94.1)	(92.5, **97.3**)
	JPEG	57.2	54.9	(49.1, 52.2, 57.5)	(51.5, 53.0, 56.7)	(44.6, 48.5, 53.6)	(48.2, 52.7, 56.9)	(46.2, 50.2, 57.7)	(56.5, **59.4**)
	Crop and rescale	30.6	32.4	(25.1, 27.8, 31.6)	(27.3, 31.6, 35.7)	(23.8, 25.3, 29.8)	(23.2, 28.8, 33.0)	(22.7, 26.0, 30.8)	(33.4, **36.8**)
	TV minimization	45.5	49.0	(40.0, 44.2, 49.7)	(43.7, 47.0, 52.2)	(43.2, 45.9, 49.8)	(39.2, 43.6, 48.5)	(41.5, 44.4, 49.3)	(48.8, **53.2**)
	Bit depth reduction	42.3	38.7	(36.5, 39.2, 45.8)	(37.1, 38.4, 42.2)	(34.5, 37.9, 41.0)	(32.9, 35.8, 39.7)	(36.1, 39.9, 43.0)	(44.6, **46.6**)
	Adversarial training	48.5	47.0	(38.6, 43.1, 49.0)	(40.5, 43.2, 48.5)	(40.9, 44.7, 51.1)	(38.7, 42.4, 47.8)	(39.2, 42.7, 48.0)	(47.6, **53.3**)
Inception-V3	No defense	81.7	84.9	(73.3, 78.6, 85.4)	(72.2, 76.7, 85.1)	(70.0, 74.4, 83.8)	(70.3, 78.8, 84.3)	(68.2, 74.6, 81.8)	(79.7, **87.2**)
	JPEG	41.8	40.5	(34.2, 38.5, 43.7)	(33.7, 36.9, 42.5)	(30.4, 34.0, 39.2)	(33.7, 37.4, 39.5)	(31.6, 34.8, 38.7)	(38.6, **44.4**)
	Crop and rescale	25.3	22.5	(18.9, 21.1, 24.7)	(19.0, 21.5, 25.9)	(15.2, 18.4, 23.0)	(16.1, 18.5, 22.8)	(16.8, 19.3, 23.7)	(22.5, **26.8**)
	TV minimization	30.8	28.2	(24.4, 28.6, 33.7)	(21.3, 26.8, 31.4)	(22.7, 25.0, 29.6)	(23.7, 26.6, 30.5)	(20.6, 23.7, 27.9)	(29.6, **33.9**)
	Bit depth reduction	36.0	33.3	(29.0, 33.6, 38.8)	(26.4, 29.2, 35.0)	(26.1, 30.6, 36.2)	(28.7, 31.3, 36.8)	(25.5, 29.0, 33.2)	(35.6, **39.2**)
	Adversarial training	35.2	32.3	(30.4, 33.2, 37.5)	(28.5, 30.9, 35.6)	(29.6, 33.1, 36.9)	(28.2, 31.4, 37.0)	(27.1, 30.4, 35.4)	(35.8, **39.9**)
SqueezeNet	No defense	87.5	89.0	(80.0, 83.7, 90.6)	(79.3, 82.2, 91.2)	(77.5, 81.8, 89.2)	(76.3, 83.2, 88.3)	(74.9, 83.2, 88.4)	(84.4, **93.3**)
	JPEG	58.3	55.8	(46.8, 53.1, 56.3)	(49.3, 54.0, 59.2)	(45.8, 48.1, 52.2)	(47.6, 49.8, 54.8)	(44.0, 47.5, 53.6)	(56.5, **61.2**)
	Crop and rescale	34.6	33.2	(27.7, 32.0, 36.8)	(26.6, 29.7, 35.9)	(26.2, 28.5, 33.7)	(24.3, 28.8, 34.0)	(25.3, 27.4, 32.1)	(34.0, **37.2**)
	TV minimization	43.0	45.1	(38.7, 42.4, 48.2)	(37.2, 42.6, **49.6**)	(34.2, 37.7, 42.0)	(36.6, 40.3, 45.7)	(36.2, 39.6, 44.7)	(45.1, 49.4)
	Bit depth reduction	46.5	44.9	(35.2, 40.4, 46.9)	(33.9, 37.3, 44.6)	(36.0, 39.2, 44.7)	(34.7, 38.2, 43.7)	(32.9, 35.0, 41.4)	(43.8, **48.4**)
	Adversarial training	42.2	43.5	(36.5, 40.3, 46.0)	(34.2, 39.6, 44.7)	(35.8, 38.7, 45.2)	(33.9, 37.4, 43.5)	(32.3, 36.4, 41.5)	(44.2, **48.8**)
ViT-B	No defense	83.7	84.4	(76.5, 80.4, 88.9)	(75.6, 79.3, 88.3)	(73.4, 78.1, 86.2)	(73.6, 80.3, 86.5)	(73.5, 76.0, 83.4)	(81.3, **89.8**)
	JPEG	49.6	50.2	(40.5, 45.0, 52.3)	(38.1, 45.8, 50.2)	(37.6, 43.5, 48.0)	(34.0, 41.4, 47.1)	(35.6, 39.3, 46.2)	(47.3, **53.6**)
	Crop and rescale	32.3	31.8	(27.8, 29.5, 34.0)	(25.2, 28.3, 33.2)	(25.0, 28.1, 32.9)	(24.4, 28.7, 31.3)	(22.2, 26.3, 31.6)	(31.2, **35.0**)
	TV minimization	40.8	41.6	(35.2, 39.7, 43.4)	(33.3, 38.0, 43.0)	(34.6, 38.2, 42.1)	(32.9, 37.3, 40.0)	(34.2, 37.9, 42.6)	(41.9, **43.5**)
	Bit depth reduction	47.3	49.2	(39.7, 43.6, 51.4)	(37.2, 44.0, 49.5)	(36.3, 41.3, 47.5)	(33.3, 39.8, 45.2)	(35.2, 39.5, 45.0)	(46.7, **52.4**)
	Adversarial training	39.4	42.0	(33.6, 35.3, 40.2)	(35.4, 38.0, 42.0)	(34.1, 38.0, 43.8)	(33.2, 35.9, 41.6)	(34.6, 36.8, 42.4)	(41.3, **44.6**)
LeViT-128	No defense	81.3	83.6	(74.2, 78.1, 87.6)	(73.5, 78.5, 85.9)	(73.8, 76.6, 84.3)	(71.2, 79.6, 84.4)	(72.5, 79.2, 85.1)	(81.1, **89.0**)
	JPEG	46.2	43.0	(39.8, 43.2, **49.0**)	(36.3, 39.8, 46.4)	(34.6, 38.2, 43.5)	(36.8, 40.1, 47.5)	(32.3, 37.6, 42.2)	(45.8, **49.0**)
	Crop and rescale	30.6	31.5	(24.0, 28.4, 33.5)	(25.6, 28.8, 35.0)	(24.6, 27.4, 31.4)	(23.2, 26.9, 30.6)	(21.8, 25.3, 29.5)	(30.2, **35.5**)
	TV minimization	36.3	33.4	(27.2, 31.9, 36.7)	(25.9, 28.0, 34.2)	(28.2, 31.9, 35.0)	(27.4, 30.5, 34.8)	(26.6, 28.9, 31.6)	(35.5, **38.2**)
	Bit depth reduction	44.5	46.2	(38.8, 43.8, 49.5)	(35.6, 38.2, 44.2)	(35.2, 39.0, 45.8)	(37.0, 40.5, 45.3)	(34.5, 37.3, 42.0)	(45.2, **51.1**)
	Adversarial training	42.5	41.7	(34.6, 39.1, 42.9)	(32.2, 37.2, 41.5)	(35.1, 40.4, 44.5)	(32.0, 36,2, 42.4)	(34.6, 38.2, 42.3)	(43.1, **47.6**)
Swin-T	No defense	85.4	87.0	(75.3, 81.4, 89.5)	(77.4, 81.6, 90.7)	(74.3, 80.6, 88.5)	(71.7, 76.0, 85.9)	(75.4, 79.7, 88.2)	(82.5, **91.3**)
	JPEG	44.2	45.7	(36.0, 40.3, 45.5)	(38.2, 41.4, 47.9)	(36.4, 39.8, 46.4)	(38.0, 43.7, 48.0)	(33.6, 36.9, 42.8)	(45.4, **49.0**)
	Crop and rescale	30.5	27.2	(24.1, 28.6, 33.4)	(22.5, 26.0, 31.2)	(21.6, 24.8, 29.5)	(22.0, 25.9, 30.2)	(21.2, 24.2, 29.0)	(30.7, **34.2**)
	TV minimization	32.8	30.4	(25.0, 28.5, 33.3)	(23.7, 26.4, 30.6)	(24.0, 28.8, 31.6)	(23.8, 27.7, 32.5)	(22.8, 25.9, 30.1)	(29.5, **33.6**)
	Bit depth reduction	43.5	40.8	(33.2, 38.6, 43.5)	(36.0, 40.3, 46.7)	(34.8, 38.2, 44.5)	(34.4, 39.7, 45.0)	(32.5, 35.3, 39.6)	(42.7, **46.9**)
	Adversarial training	34.2	33.0	(28.8, 31.5, 36.2)	(29.7, 33.1, 37.3)	(26.1, 29.6, 34.8)	(25.6, 29.0, 33.5)	(29.1, 32.4, 36.9)	(35.2, **38.7**)
MaxViT-T	No defense	87.1	88.6	(78.3, 83.7, 89.2)	(79.8, 82.7, 87.2)	(72.4, 77.7, 86.9)	(76.3, 82.0, 88.9)	(72.5, 77.0, 87.7)	(85.0, **90.8**)
	JPEG	52.5	53.8	(45.6, 49.3, 54.7)	(47.0, 50.4, 55.8)	(43.5, 47.2, 52.5)	(44.9, 48.6, 53.0)	(43.1, 46.8, 51.2)	(52.0, **56.2**)
	Crop and rescale	33.7	35.7	(26.3, 30.4, 35.9)	(26.8, 31.5, **37.7**)	(24.0, 28.2, 33.5)	(26.3, 29.4, 35.2)	(23.6, 27.9, 32.0)	(33.2, 36.8)
	TV minimization	30.4	32.8	(24.0, 28.8, 33.6)	(26.9, 29.8, 34.2)	(22.4, 25.8, 30.2)	(25.8, 29.5, **34.9**)	(21.7, 25.4, 30.7)	(31.6, **34.9**)
	Bit depth reduction	42.9	43.0	(32.6, 36.0, 41.8)	(35.6, 39.2. 44.0)	(35.2, 38.4, 42.6)	(34.2, 38.3, 42.0)	(32.7, 37.1, 41.1)	(42.4, **45.8**)
	Adversarial training	39.3	41.3	(33.6, 36.3, 40.2)	(35.3, 39.3, 43.2)	(31.4, 35.0, 38.5)	(32.0, 35.9, 39.6)	(32.2, 36.6, 40.5)	(42.2, **46.5**)

D. Attack Effectiveness Against Defense (Q3)

Here, we answer Q3 by investigating the attack effectiveness of the compared attack methods against different types of defense. In particular, we consider four image-transformation-based defense methods presented in [53], including JPEG, crop and rescale, total variance (TV) minimization, and bit depth reduction. Each defense method is applied on every perturbed image before feeding the perturbed image to the victim DNNs. We adopt the default hyperparameters of these defenses used in [53]. In addition, we also consider the adversarial training defense method [21], which aims to train a victim DNN to be more robust to attacks. Specifically, for each attack method, we follow the procedure presented in [21] to alternatively update the victim DNN and the attack model of the attack method in each iteration of the adversarial training. We adopt the default hyperparameters used in [21].

Table IV reports the FR of all the attack methods against different defense methods, where we follow the experimental settings in Section VI-B and use $\xi = 6$. We can see that compared with when no defenses are in place, the FR of all the attack methods drops when a defense is applied. Nevertheless, our boosting attack methods still consistently outperform the single-UAP methods and the highest FR is achieved by HeBA2 in most cases.

We explain the superior attack effectiveness of our boosting attack methods under different defenses as follows. *First,* UAPs often remain effective against these defenses because they generally exploit non-robust, high-dimensional features that victim DNNs inherently rely on for classification [13], [54]. These features are highly predictive, but subtle and not aligned with human perception. For the image-transformation-based defense methods, they often make local or low-level changes and do not specifically identify and remove these non-robust features [53]. For the adversarial training defense method, it often cannot explore the entire high-dimensional space of the non-robust features due to its high computational cost [55], [56], thereby leaving sufficient exploitable features for UAPs to leverage. *Second,* the key mechanism of our boosting attack framework still works in the presence of these defenses. Although the effectiveness of each of the diversified UAPs is reduced by the defenses, the sets of images successfully attacked by the diversified UAPs are still different and complementary to each other. Hence, the boosting attack framework can still improve the attack effectiveness of the single-UAP methods, thereby resulting in a higher FR of the boosting attack methods. *In summary,* the proposed boosting attack methods can maintain high FR even under these defenses because UAPs are resistant to the defenses to

TABLE V
THE AAT (MILLISECONDS) OF DIFFERENT ATTACK METHODS WHEN $\xi = 6$.

	GAP	TDA	(UAT, HoBA1, HoBA2)	(DF, HoBA1, HoBA2)	(COS, HoBA1, HoBA2)	(NAG, HoBA1, HoBA2)	(TRM, HoBA1, HoBA2)	(HeBA1, HeBA2)
ResNet-50	10.1	10.1	(<0.1, 2.3, 2.3)	(<0.1, 2.4, 2.3)	(<0.1, 2.3, 2.3)	(<0.1, 2.3, 2.3)	(<0.1, 2.2, 2.2)	(2.2, 2.2)
VGG-16	10.2	10.1	(<0.1, 2.4, 2.3)	(<0.1, 2.4, 2.3)	(<0.1, 2.4, 2.4)	(<0.1, 2.4, 2.4)	(<0.1, 2.2, 2.4)	(2.3, 2.2)
Inception-V3	10.2	10.1	(<0.1, 2.3, 2.4)	(<0.1, 2.5, 2.3)	(<0.1, 2.4, 2.4)	(<0.1, 2.3, 2.4)	(<0.1, 2.4, 2.3)	(2.4, 2.3)
SqueezeNet	10.1	10.1	(<0.1, 2.3, 2.3)	(<0.1, 2.3, 2.3)	(<0.1, 2.4, 2.3)	(<0.1, 2.4, 2.4)	(<0.1, 2.4, 2.4)	(2.4, 2.4)
ViT-B	10.3	10.2	(<0.1, 2.2, 2.3)	(<0.1, 2.3, 2.3)	(<0.1, 2.3, 2.3)	(<0.1, 2.3, 2.2)	(<0.1, 2.2, 2.2)	(2.3, 2.3)
LeViT-128	10.1	10.2	(<0.1, 2.3, 2.4)	(<0.1, 2.4, 2.4)	(<0.1, 2.4, 2.5)	(<0.1, 2.4, 2.4)	(<0.1, 2.3, 2.3)	(2.3, 2.3)
Swin-T	10.2	10.2	(<0.1, 2.2, 2.4)	(<0.1, 2.4, 2.4)	(<0.1, 2.4, 2.4)	(<0.1, 2.3, 2.4)	(<0.1, 2.3, 2.4)	(2.3, 2.4)
MaxViT-T	10.3	10.2	(<0.1, 2.3, 2.3)	(<0.1, 2.4, 2.3)	(<0.1, 2.3, 2.4)	(<0.1, 2.4, 2.4)	(<0.1, 2.3, 2.3)	(2.3, 2.3)

some extent and our boosting attack framework is effective in further improving the attack effectiveness by leveraging multiple diversified UAPs.

E. Time Cost of Attacks (Q4)

In this subsection, we answer Q4 by analyzing the attack efficiency of different attack methods. Table V reports the AAT of all the attack methods when conducting the previous experiments in Section VI-B and using $\xi = 6$. We have the following observations.

The AAT of all the compared methods is less than 10.3 milliseconds. This demonstrates the superior speed of universal adversarial attacks and image-dependent generator-based methods in launching fast attacks.

The AAT of all the single-UAP methods is less than 0.1 milliseconds since they produce a perturbed image by simply adding the UAP to the target image. However, the fast attack speed of the single-UAP methods is gained at the cost of only using a single UAP to attack all target images, which has been demonstrated to be the major bottleneck of their attack effectiveness by the previous experiments in Tables II-IV.

The AAT of the proposed boosting attack methods is about 2.4 milliseconds. Compared with the single-UAP methods, the major overhead of HoBA and HeBA methods is caused by calling the selective neural network g_θ to make a selection in the small set of diversified UAPs. Since g_θ is a lightweight SqueezeNet [46] and making the selection only needs a forward pass through g_θ, the additional time cost induced by calling g_θ is only about 2.4 milliseconds. Considering the significant improvement of attack effectiveness achieved by our methods in Tables II-IV, an additional time cost of 2.4 milliseconds is a good tradeoff when conducting fast attacks.

The AAT of GAP and TDA is around 10.1 milliseconds, which is slightly higher than that of HoBA and HeBA methods. The major overhead of GAP and TDA is caused by the complicated model architecture of the DNN-based generator that maps a target image into a perturbation.

F. The Diversity of UAPs: A Case Study (Q5)

In this subsection, we answer Q5 by conducting a case study in Fig. 2 to analyze the diversity of UAPs trained by HoBA and HeBA methods when attacking ResNet-50. We follow the experimental settings in Section VI-B and use $\xi = 6$.

Each of HoBA and HeBA methods produces a set of UAPs P, and we measure the diversity of these UAPs by the *average diversity (AD)*, that is,

$$\text{AD} = \frac{1}{\binom{K}{2}} \sum_{i \neq j} DIV_{i,j}, \tag{7}$$

where

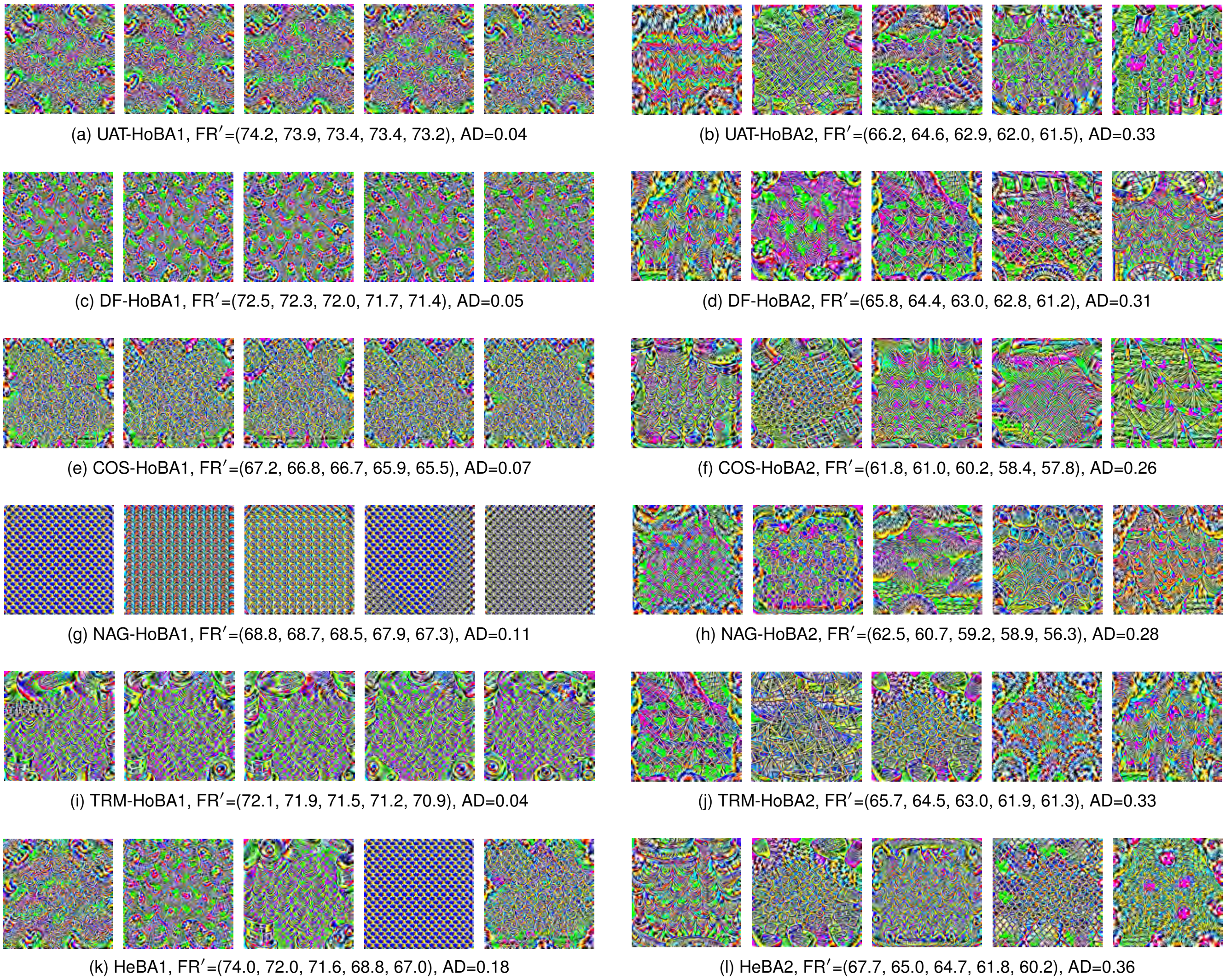
$$DIV_{i,j} = 1 - \frac{|V_i \cap V_j|}{|V_i \cup V_j|} \tag{8}$$

measures the diversity between the pair of UAPs δ_i and δ_j in P on the testing dataset D_3. V_i and V_j are the sets of successfully attacked images when using δ_i and δ_j to attack all the images in D_3, respectively. A larger $DIV_{i,j}$ indicates a greater difference between V_i and V_j, thus suggesting a higher diversity between δ_i and δ_j. Therefore, a larger AD means a higher average diversity for the UAPs in P.

Fig. 2 shows the appearance of every UAP in P, which is visualized in the same way as in [6], [24]. In the caption of each subfigure, we report the AD of the UAPs in P. In addition, we also report the fooling ratio of each UAP when applying it to attack all the images in D_3. We use the notation FR′ here to distinguish from the FR that our boosting attack methods achieve by using all the UAPs in P.

As shown in Figs. 2(a), 2(c), 2(e), and 2(i), each set of UAPs produced by HoBA1 show similar visual patterns and they have a small AD. Such results are also observed in some previous works [6], [22], [24], which indicated that multiple independent runs of a single-UAP method usually generate similar UAPs that tend to successfully attack similar sets of images. This is the major cause for the small AD. An exception is Fig. 2(g), where the UAPs generated by NAG-HoBA1 show a similar grid-like global pattern but they have different local patterns. This is because NAG samples UAPs from a trained generator, which improves the diversity of the sampled UAPs [22]. Thus, NAG-HoBA1 achieves a higher AD compared to the other HoBA1 methods.

Comparing the results of HoBA1 and HoBA2 in Fig. 2 when they are applied on the same single-UAP method, we can see that the UAPs generated by HoBA2 are more diversified than those generated by HoBA1, as shown by the appearances of the UAPs and also demonstrated by the higher AD of the UAPs generated by HoBA2. This is because HoBA2 generates the set of UAPs by training each UAP to attack a separate cluster of images. Meanwhile, due to the high AD of these UAPs, each of them often successfully attacks a large number of new images that cannot be successfully attacked by the other UAPs in P. When using this set of diversified UAPs to conduct the proposed boosting attack, the union of images successfully attacked by the UAPs of HoBA2 is much larger than that of HoBA1, as illustrated by the comparison of the Venn diagrams in Figs. 1(b.1) and 1(b.2). Therefore, we can observe in Tables II-IV that HoBA2 consistently outperforms HoBA1 in achieving higher attack effectiveness. In addition, we can also see in Fig. 2 that the FR′ of a UAP generated by



(a) UAT-HoBA1, FR′=(74.2, 73.9, 73.4, 73.4, 73.2), AD=0.04

(b) UAT-HoBA2, FR′=(66.2, 64.6, 62.9, 62.0, 61.5), AD=0.33

(c) DF-HoBA1, FR′=(72.5, 72.3, 72.0, 71.7, 71.4), AD=0.05

(d) DF-HoBA2, FR′=(65.8, 64.4, 63.0, 62.8, 61.2), AD=0.31

(e) COS-HoBA1, FR′=(67.2, 66.8, 66.7, 65.9, 65.5), AD=0.07

(f) COS-HoBA2, FR′=(61.8, 61.0, 60.2, 58.4, 57.8), AD=0.26

(g) NAG-HoBA1, FR′=(68.8, 68.7, 68.5, 67.9, 67.3), AD=0.11

(h) NAG-HoBA2, FR′=(62.5, 60.7, 59.2, 58.9, 56.3), AD=0.28

(i) TRM-HoBA1, FR′=(72.1, 71.9, 71.5, 71.2, 70.9), AD=0.04

(j) TRM-HoBA2, FR′=(65.7, 64.5, 63.0, 61.9, 61.3), AD=0.33

(k) HeBA1, FR′=(74.0, 72.0, 71.6, 68.8, 67.0), AD=0.18

(l) HeBA2, FR′=(67.7, 65.0, 64.7, 61.8, 60.2), AD=0.36

Fig. 2. The UAPs generated by the proposed boosting attack methods on ResNet-50 when $\xi = 6$. (a), (c), (e), (g) and (i) show the five UAPs produced by HoBA1. (b), (d), (f), (h) and (j) show the five UAPs produced by HoBA2. (k) and (l) show the five UAPs produced by HeBA1 and HeBA2, respectively. AD is the average diversity of the five UAPs. FR′ shows the fooling ratio of the five UAPs in the same subfigure when applying each of them to attack all images in the testing dataset D_3.

HoBA1 is often slightly larger than that of a UAP generated by HoBA2. This is because each UAP generated by HoBA1 is trained on the entire training dataset, but a UAP generated by HoBA2 is trained on a cluster of images, which contains fewer training images.

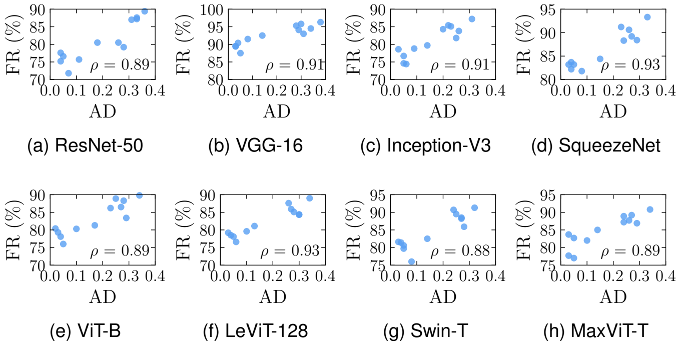
We can observe from Fig. 2 that the UAPs produced by HeBA1 and HeBA2 are more diversified than those generated by HoBA1 and HoBA2, respectively. HeBA1 achieves a higher AD than that of HoBA1, as it uses multiple different attack functions to train the set of UAPs. However, since each UAP of HeBA1 is trained on the entire training dataset, the AD of HeBA1 is still lower than that of HoBA2 and HeBA2. In addition, HeBA2 achieves the highest AD among all the boosting attack methods since it uses multiple different attack functions to train the set of UAPs and each UAP is trained on a separate cluster of training images. Due to its high AD, we can observe in Tables II-IV that HeBA2 consistently achieves the best attack effectiveness in all cases. This further demonstrates that when using the same number of UAPs, enhancing the diversity of UAPs is the key to improving the attack effectiveness of the proposed boosting attack framework, as illustrated by the comparison of the Venn diagrams in Figs. 1(b.2) and 1(b.3).

G. Correlation Analysis Between FR and AD (Q6)

In this subsection, we answer Q6 by delving more deeply into the relationship between the diversity of UAPs and the attack effectiveness of the proposed boosting attack framework.

Specifically, we conduct a correlation analysis between FR and AD in Fig. 3, where we follow the experimental settings in Section VI-B and use $\xi = 6$. Each subfigure in Fig. 3 shows the results when attacking one victim DNN. For each boosting attack method, we produce one pair of FR and AD, thus drawing one point in the subfigure. We also report the Pearson correlation coefficient (PCC) ρ computed between the FR and AD of all the points in the subfigure.

As shown in Fig. 3, points with a higher AD tend to also have a higher FR, suggesting that FR and AD exhibit a high



(a) ResNet-50 (b) VGG-16 (c) Inception-V3 (d) SqueezeNet

(e) ViT-B (f) LeViT-128 (g) Swin-T (h) MaxViT-T

Fig. 3. The FR and AD of the proposed boosting attack methods when $\xi = 6$. ρ in each subfigure is the Pearson correlation coefficient (PCC) computed between the FR and AD of all the points in the subfigure.

positive correlation. This is also evidenced by the high ρ shown in each subfigure, with the lowest ρ reaching 0.88. Such results indicate that the diversity of UAPs largely correlates with the attack effectiveness of the boosting attack framework when using the same number of UAPs. Therefore, enhancing the diversity of UAPs underpins the improvement of the attack effectiveness, which also explains the great attack effectiveness of the strongest version of our boosting attack method, that is, HeBA2.

H. Hyperparameter Analysis (Q7)

In this subsection, we answer Q7 by investigating the effect of the number of UAPs on the attack effectiveness of our boosting attack methods. We follow the experimental settings in Section VI-B and use $\xi = 6$. The other experimental details are as follows.

For HoBA methods, we use $K \in \{1, 5, 10, 50, 100\}$. When $K \in \{1, 5, 10\}$, we set the batch size to $\lfloor \frac{64}{K} \rfloor$ and set the number of training epochs to 100. However, when $K = 50$, the maximum batch size computed by $\lfloor \frac{64}{K} \rfloor$ is 1, which leads to extremely long training time; and when $K = 100$, the maximum batch size computed by $\lfloor \frac{64}{K} \rfloor$ is less than 1 and thus we cannot train the attack models of HoBA methods. To address these training issues, when $K \in \{50, 100\}$, we apply the Gumbel softmax trick to solve the HoBA-opt problem in (1). This is achieved by first following the same two steps described in Section V to convert the HoBA-opt problem to

$$\begin{aligned} \min_{P,\theta} \quad & \sum_{i=1}^{N} \mathbb{E}_{\epsilon \sim Gumbel(0,1)} \big[L(x_i, \delta_1 s_1 + \ldots + \delta_K s_K) \big], \\ \text{s.t.} \quad & \|\delta_j\|_\infty \leq \xi, \forall j \in \{1, \ldots, K\}, \end{aligned} \tag{9}$$

where s is a K-dimensional vector with the j-th entry defined as

$$s_j = \frac{\exp\big((\log \mathbb{P}_\theta(x_i)_j + \epsilon_j) / \tau \big)}{\sum_{a=1}^{K} \exp\big((\log \mathbb{P}_\theta(x_i)_a + \epsilon_a) / \tau \big)}, \tag{10}$$

and ϵ is a K-dimensional vector with each entry ϵ_j being an independent random variable following the $Gumbel(0, 1)$ distribution [39]. Then, the optimization problem in (9) can be solved following the same steps in Algorithm 3. In this way, when computing the loss $\mathbb{E}_{\epsilon \sim Gumbel(0,1)}[\cdot]$ for each training image x_i, we only need to compute the value of the attack function $L(x_i, \delta_1 s_1 + \ldots + \delta_K s_K)$ for one time by adding $\delta_1 s_1 + \ldots + \delta_K s_K$ to x_i and then feeding the perturbed image to the victim DNN. Thus, the number of perturbed images fed to the victim DNN in one training iteration is independent of the value of K and equal to the batch size. Since our GPU can accommodate up to 64 perturbed images at the same time, for $K \in \{50, 100\}$, we set the batch size to 64 and set the number of training epochs to 200.

For HeBA methods, we use $H = 5$ and $R \in \{1, 2, 10, 20\}$, meaning the total number of UAPs used by HeBA methods is $H \times R \in \{5, 10, 50, 100\}$. We set the batch size to $\lfloor \frac{64}{H} \rfloor$, using 100 training epochs when $R \in \{1, 2\}$ and 200 training epochs when $R \in \{10, 20\}$.

Fig. 4 shows the results of FR when using different numbers of UAPs, where the caption of each subfigure shows the names of the victim DNN and the single-UAP method. From the results, we can observe that the FR of all the boosting attack methods improves when using more UAPs. Such improvement is significant for HoBA2 when $K \leq 5$ and for HeBA2 when $H \times R \leq 10$. This is because the UAPs generated by HoBA2 and HeBA2 have good diversity, thus each newly added UAP will successfully attack more images that cannot be successfully attacked by the previous set of UAPs, as illustrated by the comparison of the Venn diagrams in Figs. 5(a) and 5(b).

However, we can also see from Fig. 4 that, when $K > 5$ for HoBA methods and $H \times R > 10$ for HeBA methods, further increasing the number of UAPs does not significantly improve FR. This can be explained by the diminishing marginal utility. That is, the set of successfully attacked images is already large when $K = 5$ for HoBA methods and $H \times R = 10$ for HeBA methods. Thus, although the UAPs generated by our methods are still diversified when using more UAPs, it is likely that any additional UAPs will redundantly target the same images, as illustrated by the comparison of the Venn diagrams in Figs. 5(b) and 5(c). As a result, the number of new images successfully attacked by adding more UAPs diminishes as the number of UAPs increases.

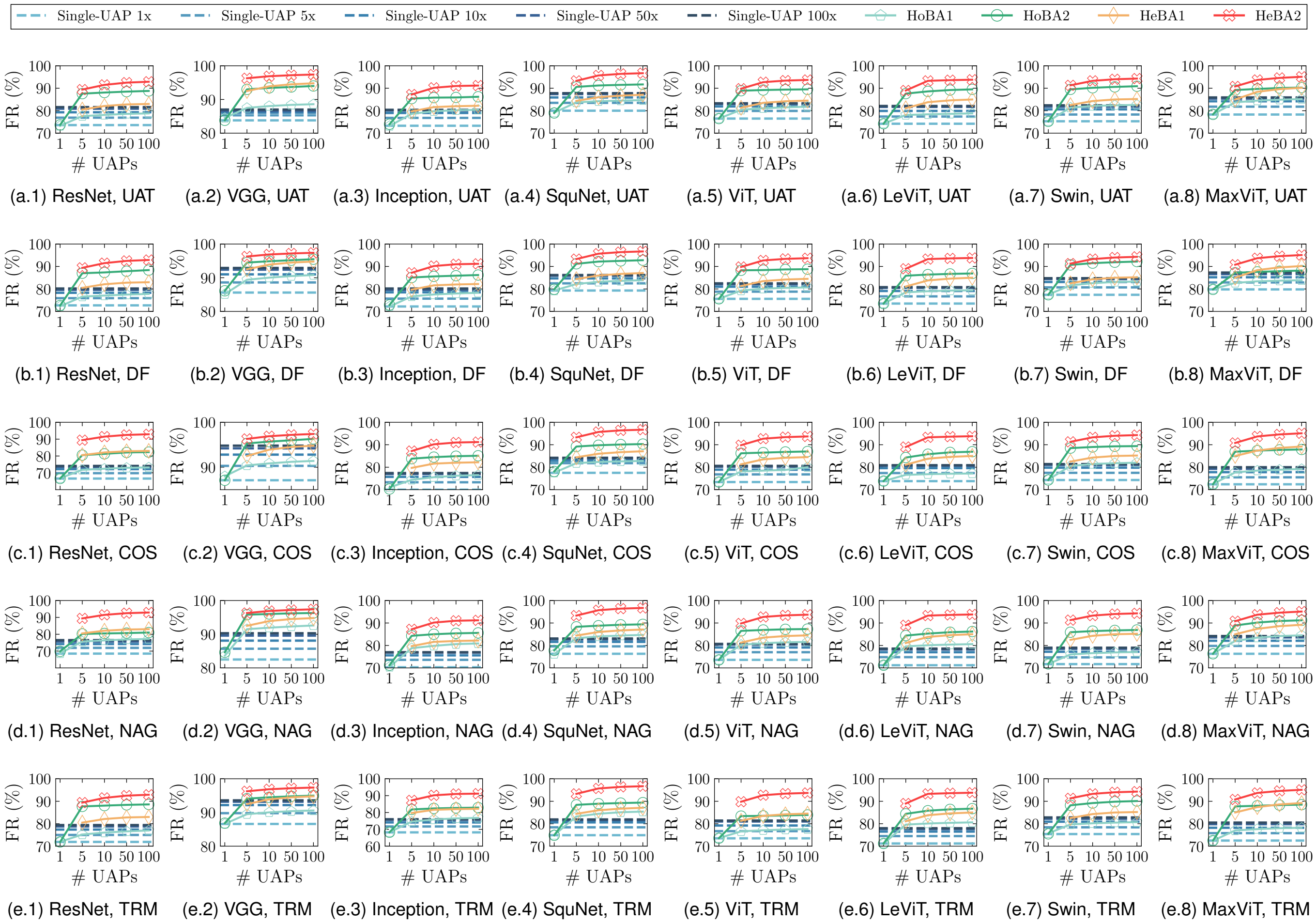
Additionally, we can see that when the number of UAPs varies, HoBA2 and HeBA2 consistently outperform their type 1 counterparts, and HeBA2 consistently achieves the highest FR among all the boosting attack methods. These results further consolidate the conclusion in Section VI-F, demonstrating that when using the same number of UAPs, generating UAPs with higher diversity will lead to higher attack effectiveness of the proposed boosting attack framework.

In summary, we can conclude from Fig. 4 that the proposed boosting attack framework can significantly improve the attack effectiveness of a single-UAP method by using only a small number of diversified UAPs.

I. Comparing to Single-UAP with More Training Images (Q8)

In this subsection, we answer Q8 by investigating how the boosting attack methods perform compared to the single-UAP methods when the single-UAP methods have more training images.

Since our boosting attack methods introduce more perturbations and the selective neural network that need to be trained, the training time cost of our methods is generally



(a.1) ResNet, UAT (a.2) VGG, UAT (a.3) Inception, UAT (a.4) SquNet, UAT (a.5) ViT, UAT (a.6) LeViT, UAT (a.7) Swin, UAT (a.8) MaxViT, UAT

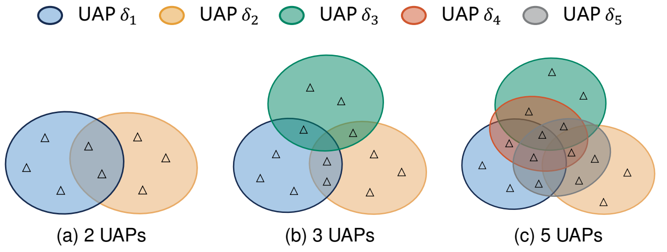
(b.1) ResNet, DF (b.2) VGG, DF (b.3) Inception, DF (b.4) SquNet, DF (b.5) ViT, DF (b.6) LeViT, DF (b.7) Swin, DF (b.8) MaxViT, DF

(c.1) ResNet, COS (c.2) VGG, COS (c.3) Inception, COS (c.4) SquNet, COS (c.5) ViT, COS (c.6) LeViT, COS (c.7) Swin, COS (c.8) MaxViT, COS

(d.1) ResNet, NAG (d.2) VGG, NAG (d.3) Inception, NAG (d.4) SquNet, NAG (d.5) ViT, NAG (d.6) LeViT, NAG (d.7) Swin, NAG (d.8) MaxViT, NAG

(e.1) ResNet, TRM (e.2) VGG, TRM (e.3) Inception, TRM (e.4) SquNet, TRM (e.5) ViT, TRM (e.6) LeViT, TRM (e.7) Swin, TRM (e.8) MaxViT, TRM

Fig. 4. The FR of the single-UAP methods and the corresponding HoBA and HeBA methods when $\xi = 6$. In each subfigure, the x-axis shows the total number of UAPs used by different boosting attack methods, which is the value of K for HoBA methods and the value of $H \times R$ for HeBA methods. The dashed horizontal lines are auxiliary lines to show the FR of the single-UAP methods when $K = 1$. The number following the name "Single-UAP" indicates that the amount of training images used for the corresponding single-UAP method is 10,000 times that number. We omit the version identifier in the name of each victim DNN and SquNet is short for SqueezeNet.



(a) 2 UAPs (b) 3 UAPs (c) 5 UAPs

Fig. 5. The Venn diagrams illustrate the sets of target images that can be successfully attacked by diversified UAPs when the number of UAPs varies. Each elliptical region in these Venn diagrams represents a set of target images in the original image space that a UAP can successfully attack. Each triangle represents a target image in the set.

higher than that of the corresponding single-UAP method. For more fair comparisons, we increase the number of training images for the single-UAP methods such that they have a similar or higher training time cost as our methods when using different numbers of UAPs. Specifically, given the number of UAPs, the number of training images for each single-UAP method is increased to the original number of training images (i.e., 10,000) multiplied by the number of UAPs. We follow the experimental setting in Section VI-H. Since the number of UAPs used in Section VI-H is in $\{1, 5, 10, 50, 100\}$, this produces single-UAP methods that use 10,000, 50,000, 100,000, 500,000 and 1,000,000 training images, which are denoted as single-UAP 1x, single-UAP 5x, single-UAP 10x, single-UAP 50x and single-UAP 100x, respectively. For all these single-UAP methods, we set the batch size to 64 and the number of training epochs to 100. Note that we still use 10,000 training images for our boosting attack methods in all the cases.

Fig. 4 shows the results of FR, from where we can observe that each single-UAP 1x method and its corresponding HoBA methods achieve similar FR at $K = 1$. This is because when $K = 1$, HoBA methods degenerate to the same single-UAP attacks as the corresponding single-UAP 1x method and their attack models are trained using the same amount of training images as that of the single-UAP 1x method.

In addition, we can also see that as the number of training images increases, the single-UAP methods achieve higher FR. This indicates that increasing the amount of training images can improve the attack effectiveness of the single-UAP methods. However, when the number of training images is larger than 100,000, adding more training images does not

TABLE VI

THE TRAINING TIME (MINUTES) OF ALL THE ATTACK METHODS ON RESNET-50 WHEN $\xi = 6$. "W.O.GUMBEL" INDICATES THAT THE HOBA-OPT PROBLEM IS SOLVED BY ALGORITHM 1 WITHOUT USING THE GUMBEL SOFTMAX TRICK. "W.GUMBEL" INDICATES THAT THE HOBA-OPT PROBLEM IS SOLVED BY APPLYING THE GUMBEL SOFTMAX TRICK.

Attack method	Single-UAP					HoBA1					HoBA2					HeBA1				HeBA2				GAP	TDA
	1x	5x	10x	50x	100x	w.o.Gumbel			w.Gumbel		w.o.Gumbel			w.Gumbel											
# UAPs	1	1	1	1	1	1	5	10	50	100	1	5	10	50	100	5	10	50	100	5	10	50	100	n/a	n/a
# Training images	10,000	50,000	100,000	500,000	1,000,000	10,000					10,000					10,000				10,000				10,000	10,000
UAT	43	197	417	2,107	4,196	95	425	855	2,258	4,356	54	230	454	123	128	648	814	2,377	4,211	287	296	568	576	235	267
DF	45	203	420	2,068	4,185	98	435	885	2,303	4,651	56	239	469	131	137										
COS	42	195	415	2,055	4,189	94	430	843	2,231	4,393	55	226	440	128	130										
NAG	166	730	1,560	7,636	13,944	221	431	683	328	345	57	232	420	139	152										
TRM	52	214	438	2,227	4,245	108	513	972	2,741	5,362	64	278	537	133	137										

significantly improve FR. This implies the inherent limitation of the single-UAP methods, that is, using a single UAP to attack a wide variety of target images has limited attack effectiveness even with more training images.

Moreover, as shown in Fig. 4, the best performing single-UAP method (i.e., single-UAP 100x) underperforms HoBA2 and HeBA2 in most cases when using only 5 UAPs. Table VI reports the training time of the single-UAP methods and the boosting attack methods when conducting the experiments in Fig. 4 on attacking ResNet-50. We can see that due to the large number of training images, the training time of the single-UAP 100x method is much higher than that of HoBA2 and HeBA2 when using 5 UAPs. These results demonstrate that with similar or even higher training time costs, the attack effectiveness of the single-UAP methods is still inferior to that of HoBA2 and HeBA2. This further highlights the superiority of our boosting attack framework in improving the attack effectiveness of the single-UAP methods by using a small set of UAPs. We will discuss the training time of all the attack methods in detail in Appendix D.

VII. Conclusion

In this paper, we propose a novel boosting attack framework to significantly improve the attack effectiveness of existing single-UAP methods while maintaining a fast attack speed. The key idea is to simultaneously train a set of diversified UAPs and a selective neural network, and use the selective neural network to choose the most effective UAP when attacking a target image. Extensive experiments show the superior performance of the proposed boosting attack framework in different attack settings and against various defenses. We also reveal that enhancing the diversity of UAPs is the key to achieving higher attack effectiveness for our framework. In future work, we will continue to explore novel ways to further enhance the diversity of UAPs.

References

[1] S. Li, X. Liao, X. Che, X. Li, Y. Zhang, and L. Chu, "Cocktail universal adversarial attack on deep neural networks," in *European Conference on Computer Vision*, 2024, pp. 396–412.

[2] Y. Zhu, W. Min, and S. Jiang, "Attribute-guided feature learning for few-shot image recognition," *IEEE Transactions on Multimedia*, vol. 23, pp. 1200–1209, 2020.

[3] T. Wu, S. Tang, R. Zhang, J. Cao, and Y. Zhang, "Cgnet: A light-weight context guided network for semantic segmentation," *IEEE Transactions on Image Processing*, vol. 30, pp. 1169–1179, 2020.

[4] J. Le, D. Zhang, X. Lei, L. Jiao, K. Zeng, and X. Liao, "Privacy-preserving federated learning with malicious clients and honest-but-curious servers," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4329–4344, 2023.

[5] J. Le, X. Lei, N. Mu, H. Zhang, K. Zeng, and X. Liao, "Federated continuous learning with broad network architecture," *IEEE Transactions on Cybernetics*, vol. 51, no. 8, pp. 3874–3888, 2021.

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1765–1773.

[7] C. Wan, F. Huang, and X. Zhao, "Average gradient-based adversarial attack," *IEEE Transactions on Multimedia*, vol. 25, pp. 9572–9585, 2023.

[8] L. Gao, Z. Huang, J. Song, Y. Yang, and H. T. Shen, "Push and pull: Transferable adversarial examples with attentive attack," *IEEE Transactions on Multimedia*, vol. 24, pp. 2329–2338, 2021.

[9] H. Gong, M. Dong, S. Ma, S. Camtepe, S. Nepal, and C. Xu, "Stealthy physical masked face recognition attack via adversarial style optimization," *IEEE Transactions on Multimedia*, vol. 26, pp. 5014–5025, 2023.

[10] X. Wang, H. Chen, P. Sun, J. Li, A. Zhang, W. Liu, and N. Jiang, "Advst: Generating unrestricted adversarial images via style transfer," *IEEE Transactions on Multimedia*, vol. 26, pp. 4846–4858, 2023.

[11] X. Wei and S. Zhao, "Boosting adversarial transferability with learnable patch-wise masks," *IEEE Transactions on Multimedia*, vol. 26, pp. 3778–3787, 2023.

[12] W. Feng, N. Xu, T. Zhang, Y. Zhang, and F. Wu, "Enhancing cross-task transferability of adversarial examples via spatial and channel attention," *IEEE Transactions on Multimedia*, 2024.

[13] M. Li, Y. Yang, K. Wei, X. Yang, and H. Huang, "Learning universal adversarial perturbation by adversarial example," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022, pp. 1350–1358.

[14] Y. Liu, X. Feng, Y. Wang, W. Yang, and D. Ming, "Trm-uap: Enhancing the transferability of data-free universal adversarial perturbation via truncated ratio maximization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4762–4771.

[15] J. Weng, Z. Luo, Z. Zhong, D. Lin, and S. Li, "Exploring non-target knowledge for improving ensemble universal adversarial attacks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, pp. 2768–2775.

[16] X. Liu, Y. Zhong, Y. Zhang, L. Qin, and W. Deng, "Enhancing generalization of universal adversarial perturbation through gradient aggregation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4435–4444.

[17] D. Wang, W. Yao, T. Jiang, and X. Chen, "Improving transferability of universal adversarial perturbation with feature disruption," *IEEE Transactions on Image Processing*, vol. 33, pp. 722–737, 2023.

[18] J. Lian, X. Wang, Y. Su, M. Ma, and S. Mei, "Cba: Contextual background attack against optical aerial detection in the physical world," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1–16, 2023.

[19] J. Lian, S. Mei, S. Zhang, and M. Ma, "Benchmarking adversarial patch against aerial detection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–16, 2022.

[20] X. Wang, S. Mei, J. Lian, and Y. Lu, "Fooling aerial detectors by background attack via dual-adversarial-induced error identification," *IEEE Transactions on Geoscience and Remote Sensing*, 2024.

[21] A. Shafahi, M. Najibi, Z. Xu, J. Dickerson, L. S. Davis, and T. Goldstein, "Universal adversarial training," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 5636–5643.

[22] K. R. Mopuri, U. Ojha, U. Garg, and R. V. Babu, "Nag: Network for adversary generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 742–751.

[23] Y. Deng and L. J. Karam, "Universal adversarial attack via enhanced projected gradient descent," in *IEEE International Conference on Image Processing*, 2020, pp. 1241–1245.

[24] C. Zhang, P. Benz, A. Karjauv, and I. S. Kweon, "Data-free universal adversarial perturbation and black-box attack," in *Proceedings of the*

IEEE/CVF International Conference on Computer Vision, 2021, pp. 7868–7877.

[25] C. Zhang, P. Benz, T. Imtiaz, and I. S. Kweon, "Understanding adversarial examples from the mutual influence of images and perturbations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 521–14 530.

[26] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy*, 2017, pp. 39–57.

[27] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *International Conference on Machine Learning*, 2018, pp. 274–283.

[28] K. K. Nakka1 and M. Salzmann, "Learning transferable adversarial perturbations," *Advances in Neural Information Processing Systems*, pp. 13 950–13 962, 2021.

[29] O. Poursaeed, I. Katsman, B. Gao, and S. Belongie, "Generative adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4422–4431.

[30] C. Zhang, P. Benz, T. Imtiaz, and I.-S. Kweon, "Cd-uap: Class discriminative universal adversarial perturbation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 6754–6761.

[31] P. Benz, C. Zhang, T. Imtiaz, and I. S. Kweon, "Double targeted universal adversarial perturbations," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[32] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, 2014.

[33] J. Wang, T. Zhang, S. Liu, P.-Y. Chen, J. Xu, M. Fardad, and B. Li, "Adversarial attack generation empowered by min-max optimization," *Advances in Neural Information Processing Systems*, pp. 16 020–16 033, 2021.

[34] C. Zhang, P. Benz, C. Lin, A. Karjauv, J. Wu, and I. S. Kweon, "A survey on universal adversarial attack," *arXiv preprint arXiv:2103.01498*, 2021.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[36] A. A. Goldstein, "Convex programming in hilbert space," *Bulletin of the American Mathematical Society*, pp. 709–710, 1964.

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

[38] J. Weng, Z. Luo, D. Lin, and S. Li, "Comparative evaluation of recent universal adversarial perturbations in image classification," *Computers and Security*, vol. 136, p. 103576, 2024.

[39] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[40] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[42] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the European Conference on Computer Vision*, 2014, pp. 740–755.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[46] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[47] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[48] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "Levit: a vision transformer in convnet's clothing for faster inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 259–12 269.

[49] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[50] Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. Bovik, and Y. Li, "Maxvit: Multi-axis vision transformer," in *European Conference on Computer Vision*, 2022, pp. 459–479.

[51] M. Tsivgoulis, T. Papastergiou, and V. Megalooikonomou, "An improved squeezenet model for the diagnosis of lung cancer in ct scans," *Machine Learning with Applications*, vol. 10, p. 100399, 2022.

[52] A. S. Agoes, Z. Hu, and N. Matsunaga, "Fine tuning based squeezenet for vehicle classification," in *Proceedings of the International Conference on Advances in Image Processing*, 2017, pp. 14–18.

[53] C. Guo, M. Rana, M. Cisse, and L. Van Der Maaten, "Countering adversarial images using input transformations," *arXiv preprint arXiv:1711.00117*, 2017.

[54] S. M. Park, K.-A. Wei, K. Xiao, J. Li, and A. Madry, "On distinctive properties of universal perturbations," *arXiv preprint arXiv:2112.15329*, 2021.

[55] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[56] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," *arXiv preprint arXiv:1805.12152*, 2018.

[57] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2010.

[58] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision*, 2016, pp. 21–37.

[59] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[60] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324.

[61] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[62] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

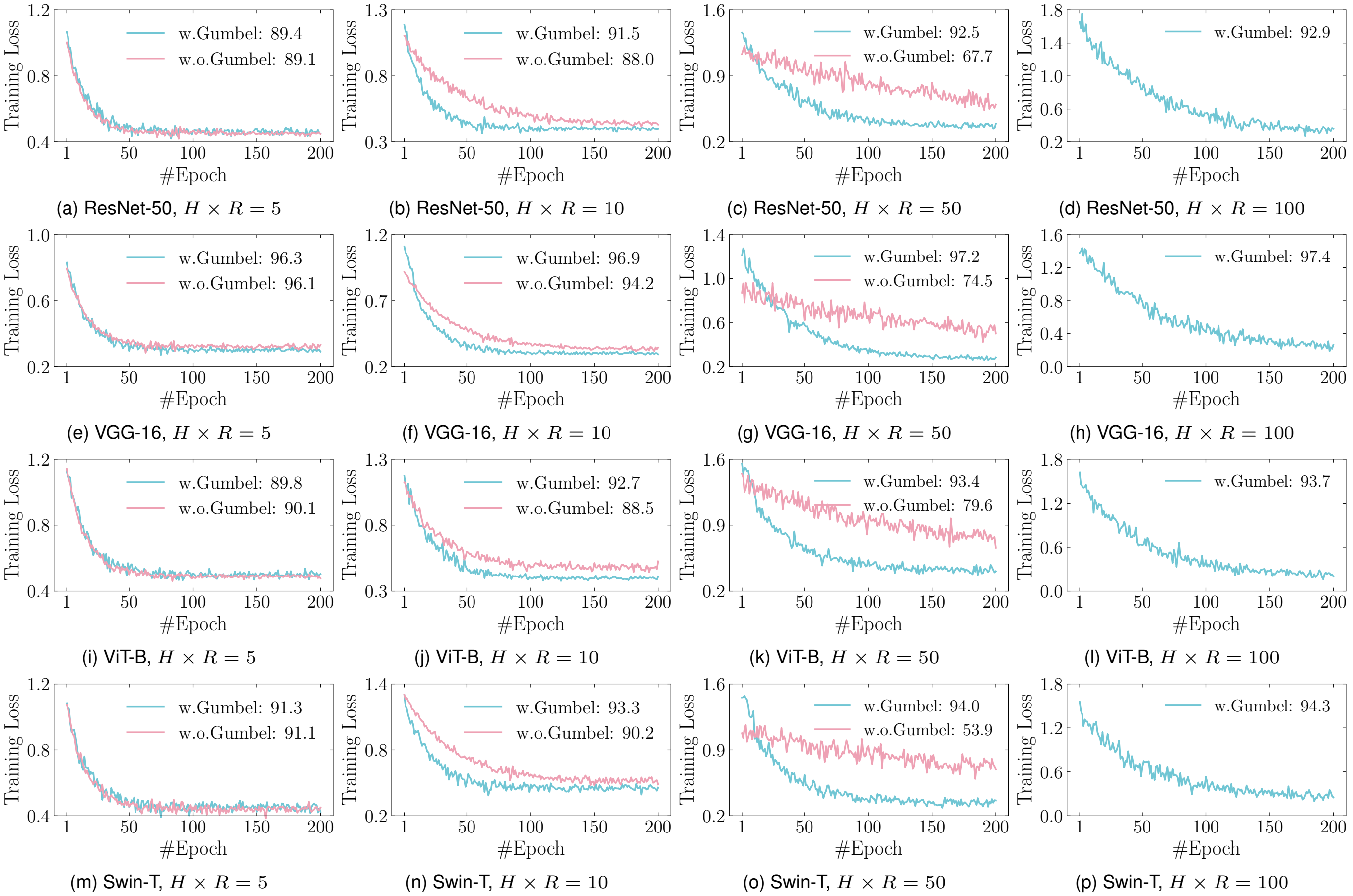
[63] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[64] A. Liu, J. Wang, X. Liu, B. Cao, C. Zhang, and H. Yu, "Bias-based universal adversarial patch attack for automatic check-out," in *Proceedings of the European Conference on Computer Vision*, 2020, pp. 395–410.

[65] X. Wei, Y. Guo, and J. Yu, "Adversarial sticker: A stealthy attack method in the physical world," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 2711–2725, 2022.

[66] A. Sharma, Y. Bian, P. Munz, and A. Narayan, "Adversarial patch attacks and defences in vision-based tasks: A survey," *arXiv preprint arXiv:2206.08304*, 2022.

[67] Y. Li, S. Bai, C. Xie, Z. Liao, X. Shen, and A. Yuille, "Regional homogeneity: Towards learning transferable universal adversarial perturbations against defenses," in *Proceedings of the European Conference on Computer Vision*, 2020, pp. 795–813.



(a) ResNet-50, $H \times R = 5$ (b) ResNet-50, $H \times R = 10$ (c) ResNet-50, $H \times R = 50$ (d) ResNet-50, $H \times R = 100$

(e) VGG-16, $H \times R = 5$ (f) VGG-16, $H \times R = 10$ (g) VGG-16, $H \times R = 50$ (h) VGG-16, $H \times R = 100$

(i) ViT-B, $H \times R = 5$ (j) ViT-B, $H \times R = 10$ (k) ViT-B, $H \times R = 50$ (l) ViT-B, $H \times R = 100$

(m) Swin-T, $H \times R = 5$ (n) Swin-T, $H \times R = 10$ (o) Swin-T, $H \times R = 50$ (p) Swin-T, $H \times R = 100$

Fig. 6. The training loss curves of w.Gumbel and w.o.Gumbel when $\xi = 6$ and $H \times R \in \{5, 10, 50, 100\}$. The numbers in the legend of each subfigure show the FR achieved by using the trained selective neural network g_θ and perturbations in P to attack images in the testing dataset D_3.

Appendix A
Comparing the Efficiency of Algorithm 1 and Algorithm 3

In this section, we investigate the effectiveness of using the Gumbel softmax trick [39] to improve the training efficiency of HeBA. In particular, we analyze the training convergence speed of HeBA when solving the HeBA-opt problem with and without using the Gumbel softmax trick. We denote by **w.Gumbel** the training method that uses the Gumbel softmax trick, which is described in Algorithm 3; and we denote by **w.o.Gumbel** the training method that does not use the Gumbel softmax trick, which directly solves the HeBA-opt problem in (2) in the same way as in Algorithm 1.

Fig. 6 shows the training loss curves of w.Gumbel and w.o.Gumbel when attacking ResNet-50, VGG-16, ViT-B and Swin-T. In our experiments, we set $\xi = 6$, $H = 5$ and $R \in \{1, 2, 10, 20\}$, thus the total number of UAPs trained by HeBA is $H \times R \in \{5, 10, 50, 100\}$. For the w.o.Gumbel method described in Algorithm 1, we set $K = H \times R$ such that it trains the same amount of UAPs as w.Gumbel.

For both w.Gumbel and w.o.Gumbel, we use the maximum batch size that can be accommodated by our GPU, that is, $\lfloor \frac{64}{K} \rfloor$ for w.o.Gumbel and $\lfloor \frac{64}{H} \rfloor$ for w.Gumbel. Since $K = H \times R$, the batch size of w.Gumbel is R times larger than that of w.o.Gumbel. The root cause of this is that w.Gumbel saves the memory cost by R times when compared to w.o.Gumbel.

We can observe from Fig. 6 that, when $R = 1$, w.Gumbel and w.o.Gumbel converge at a comparable speed. This is because $K = H$ when $R = 1$, and therefore w.Gumbel and w.o.Gumbel use the same batch size. However, when $R > 1$, we can see that w.Gumbel converges much faster than w.o.Gumbel as R increases. This is because w.Gumbel uses a larger batch size than w.o.Gumbel when $R > 1$.

We could not report the results of w.o.Gumbel when $R = 20$ in Fig. 6, since the maximum batch size of w.o.Gumbel, computed by $\lfloor \frac{64}{5*20} \rfloor$, is smaller than 1. This means our GPU memory is not large enough to accommodate a batch size of 1 for w.o.Gumbel. However, w.Gumbel still trains efficiently when $R = 20$, because its memory cost per batch is independent of R.

The numbers in the legend of each subfigure in Fig. 6 show the FR of w.Gumbel and w.o.Gumbel, respectively. The FR is computed by using the attack model to attack images in the testing dataset D_3. The attack model trained by w.Gumbel achieves comparable FR to that trained by w.o.Gumbel when $R = 1$. However, when $R > 1$, w.Gumbel achieves a higher FR than w.o.Gumbel. This is because w.o.Gumbel is struggling to converge due to the small batch size limited by its high consumption of GPU memory.

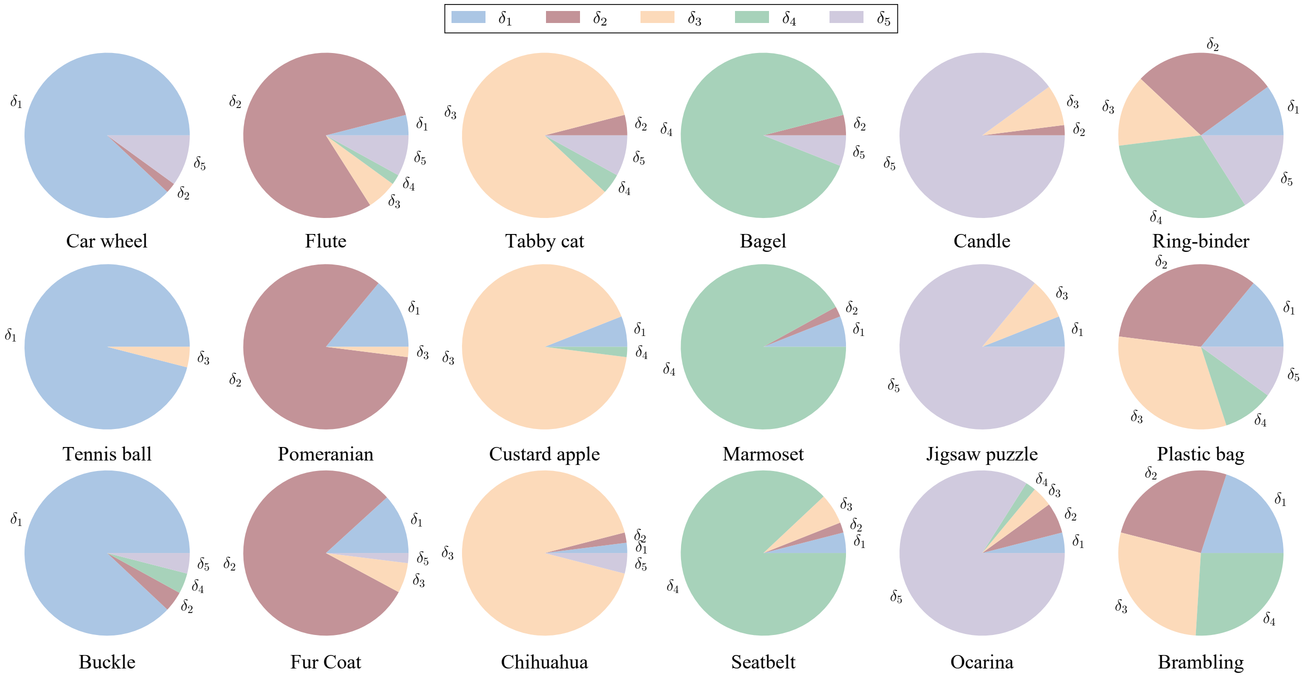


Fig. 7. The proportion of images in the same class attacked by each of the diversified UAPs. $\delta_1, \ldots, \delta_5$ are the five UAPs in Fig. 2(l) (from left to right). The name under each pie chart is the name of an image class. The area of each colored region shows the proportion of images attacked by one UAP.

Appendix B
The Distribution of Images Attacked by Different UAPs (Q9)

In this section, we answer Q9 by using the five diversified UAPs generated by HeBA2 as an example to investigate the distribution of images attacked by each of the diversified UAPs. Each pie chart in Fig. 7 shows the results for a specific class. In each pie chart, the five regions show the proportion of images attacked by the five diversified UAPs $\delta_1, \ldots, \delta_5$, which are visualized in Fig. 2(l) from left to right. The 1,000 classes of images in our testing dataset D_3 produce 1,000 pie charts, which cannot be shown in the paper due to the limited space. However, we find that these classes generally fall into six typical categories, thus we give some examples in each category in Fig. 7.

The first category of classes, such as “Car wheel”, “Tennis ball” and “Buckle”, is shown in the first column of Fig. 7. For these classes, the UAP δ_1 is selected by the selective neural network g_θ to attack most of the images. Such classes are dominated by δ_1, because δ_1 is likely the most effective in attacking the images in these classes. Similarly, the other four categories of classes shown in the second to fifth columns of Fig. 7 are dominated by δ_2, δ_3, δ_4 and δ_5, respectively. The sixth category of classes shown in the last column of Fig. 7 is not significantly dominated by any UAP, because the proportions of images attacked by each of the five UAPs tend to be balanced. This is because all the UAPs are comparably effective in attacking the images in these classes, thus the selective neural network g_θ makes a more balanced selection when attacking images in such classes.

The above results demonstrate an interesting class-specific property of the diversified UAPs trained by our boosting attack framework. That is, each diversified UAP is highly effective in dominating the attack of certain classes, and the sets of classes dominated by different UAPs are different. Such a property is the key to the great performance of the proposed boosting attack framework, because, when the selective neural network g_θ chooses the most effective UAP to attack each new image, the final set of successfully attacked images will be the union of the images successfully attacked by every UAP.

Appendix C
The Impact of Different Model Architectures of the Selective Neural Network g_θ (Q10)

In this section, we answer Q10 by investigating the impact of different model architectures of the selective neural network g_θ on the performance of our boosting attack methods. In particular, we focus on HeBA2 since it generally performs the best out of all the boosting attack methods, as shown in Tables II-IV and Fig. 4.

Tables VII and VIII report the results of FR and AAT, respectively, when using SqueezeNet, ResNet-50, VGG-16 and Inception-V3 as the model architectures of the selective neural network g_θ. We follow the experimental settings in Section VI-B and use $\xi = 6$. For each of the model architectures, we adopt its original architecture and use $H \times R$ softmax-activated output neurons in the last layer.

As shown in Table VII, using different model architectures for g_θ does not have a significant effect on the FR of HeBA2. This demonstrates that the attack effectiveness of HeBA2 is not very sensitive to the choice of g_θ and HeBA2 can consistently achieve high attack effectiveness because of the boosting mechanism used in the proposed boosting attack framework. However, as shown in Table VIII, choosing SqueezeNet as g_θ consistently results in a lower AAT due to its lightweight

TABLE VII
THE FR (%) OF HEBA2 WHEN USING DIFFERENT MODEL ARCHITECTURES FOR THE SELECTIVE NEURAL NETWORK g_θ.

Victim DNN	Selective neural network g_θ			
	SqueezeNet	ResNet-50	VGG-16	Inception-V3
ResNet-50	89.4	89.8	88.9	88.7
VGG-16	96.3	96.0	95.7	95.8
Inception-V3	87.2	87.4	87.0	86.6
SqueezeNet	93.3	92.9	93.5	92.7
ViT-B	89.8	89.5	90.3	89.6
LeViT-128	89.0	89.9	88.7	88.4
Swin-T	91.3	91.4	90.7	89.7
MaxViT-T	90.8	90.7	90.4	90.9
Average	90.9	91.0	90.7	90.3

TABLE VIII
THE AAT (MILLISECONDS) OF HEBA2 WHEN USING DIFFERENT MODEL ARCHITECTURES FOR THE SELECTIVE NEURAL NETWORK g_θ.

Victim DNN	Selective neural network g_θ			
	SqueezeNet	ResNet-50	VGG-16	Inception-V3
ResNet-50	2.2	9.0	24.2	12.7
VGG-16	2.2	9.1	24.3	12.7
Inception-V3	2.3	9.1	24.2	12.6
SqueezeNet	2.4	9.0	24.1	12.7
ViT-B	2.3	9.1	24.2	12.7
LeViT-128	2.3	9.0	24.3	12.8
Swin-T	2.4	9.0	24.2	12.7
MaxViT-T	2.3	9.0	24.2	12.7
Average	2.3	9.0	24.2	12.7

network structure. Therefore, we use SqueezeNet as the default model architecture of g_θ.

APPENDIX D
THE TRAINING TIME OF ATTACK METHODS (Q11)

In this section, we answer Q11 by discussing the training time of the proposed boosting attack methods and the baseline methods reported in Table VI. The training time refers to the time cost used by each method to train its attack model in an offline manner. It does not include the time cost to conduct an attack on a new image.

For the single-UAP methods, Table VI reports their training time when using different numbers of training images. For HoBA methods, Table VI reports their training time when $K \in \{1, 5, 10, 50, 100\}$. Thus, the "# UAPs" shows the value of K for each HoBA method. For HeBA methods, Table VI reports their training time when $H = 5$ and $R \in \{1, 2, 10, 20\}$, which means the total number of UAPs used by HeBA methods is $H \times R \in \{5, 10, 50, 100\}$. Thus, the "# UAPs" shows the value of $H \times R$ for each HeBA method. Additionally, Table VI also reports in the last two columns the training time of GAP and TDA when conducting the experiments in Section VI-B. Since GAP and TDA train a DNN-based generator to produce an image-dependent perturbation for each target image, their training time is the time cost to train the generator. Thus, the "# UAPs" is not applicable for GAP and TDA.

Next, we analyze and discuss the reported results in the following four parts.

Part 1: the training time of the single-UAP methods.

The training time of each single-UAP method is approximately the training time of the corresponding single-UAP 1x method multiplied by the ratio between the number of training images of the single-UAP method and that of the single-UAP 1x method. We explain the reason as follows. Denoted by r this ratio and by I the total number of training iterations of the single-UAP 1x method. Since the number of training images for each single-UAP method is r times that of the single-UAP 1x method, and all the single-UAP methods use the same batch size and the same number of training epochs, the total number of training iterations for each single-UAP method is $I \times r$. Since each training iteration costs a similar amount of time, the training time of each single-UAP is about r times that of the single-UAP 1x method.

Part 2: the training time of HoBA methods.

For both HoBA1 and HoBA2, when $K \in \{1, 5, 10\}$, we solve the HoBA-opt problem in (1) by Algorithm 1, which was proposed in our conference paper previously published in ECCV 2024 [1]. However, since the largest batch size used by Algorithm 1 is $\lfloor \frac{64}{K} \rfloor$, the computed batch size is too small when $K \in \{50, 100\}$, which hinders the training of the attack models for HoBA1 and HoBA2. Therefore, in our journal paper, we apply the Gumbel softmax trick to solve the HoBA-opt problem to accommodate $K \in \{50, 100\}$ for HoBA1 and HoBA2. The implementation details were previously described in Section VI-H.

As reported in Table VI, the training time of HoBA2 is approximately K times that of the corresponding single-UAP 1x method when $K \in \{1, 5, 10\}$. This phenomenon is caused by the different batch sizes used by the single-UAP 1x method and HoBA2 when solving the HoBA-opt problem by Algorithm 1 without using the Gumbel softmax trick. For both methods, we use the largest batch size that consumes all the GPU memory. This leads to a batch size of 64 for the single-UAP 1x method and a batch size of $\lfloor \frac{64}{K} \rfloor$ for HoBA2. As a result, the number of training iterations in each training epoch of HoBA2 is K times that of the single-UAP 1x method. Since HoBA2 and the single-UAP 1x method use the same number of training epochs when $K \in \{1, 5, 10\}$, the training time of HoBA2 is approximately K times that of the single-UAP 1x method.

However, when $K \in \{50, 100\}$, the training time of HoBA2 is much smaller than K times that of the corresponding single-UAP 1x method. This is because HoBA2 applies the Gumbel softmax trick to solve the HoBA-opt problem when $K \in \{50, 100\}$. As explained in Section VI-H, using the Gumbel softmax trick allows HoBA2 to consistently use a batch size of 64 for different values of K. Therefore, when $K \in \{50, 100\}$, the number of training iterations in each training epoch of HoBA2 is the same as that of the single-UAP 1x method, which reduces the training time of HoBA2. Nevertheless, since the number of training epochs is set to 200 for HoBA2 when $K \in \{50, 100\}$ and to 100 for the single-UAP 1x method, the training time of HoBA2 is still higher than that of the single-UAP 1x method.

We can also see from Table VI that the training time of HoBA1 is about $2K$ times that of the corresponding single-UAP 1x method when $K \in \{1, 5, 10\}$. We explain this as follows. Denote by E the training time of the single-UAP 1x method, HoBA1 first trains K UAPs by independently running the single-UAP 1x method K times, which costs $K \times E$ time. Then, HoBA1 takes another $K \times E$ time to solve the HoBA-opt problem by Algorithm 1 without using the Gumbel softmax trick. Therefore, the overall training time of HoBA1 is about $2K \times E$ when $K \in \{1, 5, 10\}$.

However, when $K \in \{50, 100\}$, the training time of HoBA1 is approximately $K \times E$. This is because after training the K UAPs, HoBA1 applies the Gumbel softmax trick to solve the HoBA-opt problem when $K \in \{50, 100\}$, which costs much less time than $K \times E$. Thus, when $K \in \{50, 100\}$, the training time of HoBA1 is mainly dominated by the time to train the K UAPs, which is $K \times E$.

One special case is the training time of NAG-HoBA1 and NAG-HoBA2, which does not align with the above analysis. For NAG-HoBA1, the root cause is that the training time of NAG is dominated by the time cost to train a DNN-based generator. NAG-HoBA1 does not independently train K DNN-based generators; instead, it only trains a single DNN-based generator and uses it to sample K UAPs. Therefore, denoted by E the training time of the NAG 1x method, the training time of NAG-HoBA1 shown in Table VI is less than $2K \times E$ when $K \in \{1, 5, 10\}$ and less than $K \times E$ when $K \in \{50, 100\}$. For NAG-HoBA2, it has a training time much less than $K \times E$ when $K \in \{1, 5, 10, 50, 100\}$. This is because the NAG 1x method costs a lot of time to train the generator; instead, NAG-HoBA2 only utilizes its fooling loss [22] as the attack function $L(x_i, \delta_j)$ to train the K UAPs and the selective neural network g_θ.

Part 3: the training time of HeBA methods.

As shown in Table VI, the training time of HeBA2 stays stable for different values of $H \times R$ when using the same number of training epochs. For the cases of $H \times R = 5$ and $H \times R = 10$, where the number of training epochs is set to 100, HeBA2 costs a similar amount of training time in both cases. For the cases of $H \times R = 50$ and $H \times R = 100$, where the number of training epochs is set to 200, HeBA2 also costs a similar amount of training time in both cases. The reason for the above phenomenon is that we set the batch size to $\lfloor \frac{64}{H} \rfloor$ when training HeBA2. Since the batch size is independent of R and we set $H = 5$ for all the experiments, the training time of HeBA2 stays stable when using the same number of training epochs.

However, we can also see that the value of $H \times R$ affects the training time of HeBA1. This is because, prior to solving the HeBA-opt problem, HeBA1 independently runs each of the five single-UAP 1x methods for R times to train R UAPs. Consequently, when $H = 5$ is set as a fixed value, a larger value of $H \times R$ means a larger value of R, which increases the training time of HeBA1.

Part 4: the training time of GAP and TDA.

As reported in Table VI, the training time of GAP and TDA is longer than that of the single-UAP 1x methods when using the same number of training images. This is because, instead of training a single UAP, they need to train a complicated DNN-based generator to produce perturbations. The time cost to train the generator dominates their training time. In addition, the training time of GAP and TDA is comparable to that of HoBA2 and HeBA2 when "# UAPs" is 5. However, as shown in Tables II-IV, when HoBA2 and HeBA2 use 5 UAPs, they achieve higher attack effectiveness than GAP and TDA in most cases. Therefore, HoBA2 and HeBA2 are more cost-effective in terms of the attack effectiveness achieved by consuming a similar amount of training time.

In summary, since UAP-based attacks, such as the single-UAP methods and the proposed HoBA and HeBA methods, are often trained offline in the literature [6], [21], [30], [34], the training time is often not a big concern for attackers. More importantly, as shown by the experimental results in Tables II-IV and Fig. 4, using a small number of 5 UAPs is sufficient for our boosting attack methods to achieve significant attack effectiveness. This means the additional offline training time of our methods is worth the effort of a hacker.

Appendix E
Attack Effectiveness in Object Detection (Q12)

In this section, we answer Q12 by investigating the attack effectiveness of the proposed boosting attack methods in the object detection task.

For each of our methods and the single-UAP methods, we follow [24] to extend it to attack a victim detector by using its attack function $L(x_i, \delta_j)$ to measure the similarity between the predictions of correctly matched objects made by the victim detector before and after the perturbation δ_j is added to the target image x_i. The baseline methods GAP and TDA can be directly applied to attack object detectors without any extension [28], [29].

We follow the experimental settings in [28]. Specifically, we adopt the training dataset of PASCAL VOC [57] to train the attack model of each attack method and adopt the testing dataset of PASCAL VOC to evaluate the attack effectiveness of different attack methods. We choose the SSD framework [58] with 4 different backbones, namely, ResNet-50 [44], VGG-16 [44], EfficientNet [59], and MobileNet-V3 [60], as the victim detectors. They are all pretrained on the training dataset of PASCAL VOC. We evaluate the attack effectiveness of an attack method against a victim detector by the *mean Average Precision (mAP)* [28] of the victim detector achieves on the testing dataset. A lower mAP indicates worse predictive performance of the victim detector, and thus implies better attack effectiveness of that attack method. We report mAP in percentage by default. The other experimental settings are the same as in Section VI-B, except that we use $\xi = 12$ for all the attack methods.

Table IX reports the results of mAP of all the attack methods. We can see that in all cases, the mAP of our boosting attack methods is lower than that of the corresponding single-UAP method. This demonstrates that the proposed boosting attack framework is also effective in improving the attack effectiveness of the single-UAP methods in the object detection task. In addition, we can also observe that the lowest mAP is consistently achieved by HeBA2 as shown by the bold numbers in each row of Table IX. Such results are consistent with those in Tables II-IV and Fig. 4, which demonstrate the great performance of HeBA2. Moreover, these results also indicate that when attacking object detectors, enhancing the

TABLE IX

THE MAP (%) WHEN $\xi = 12$. BOLD NUMBER MARKS THE LOWEST MAP IN EACH ROW. UNDERLINED NUMBER SHOWS THE RUNNER UP PERFORMANCE.

	GAP	TDA	(UAT, HoBA1, HoBA2)	(DF, HoBA1, HoBA2)	(COS, HoBA1, HoBA2)	(NAG, HoBA1, HoBA2)	(TRM, HoBA1, HoBA2)	(HeBA1, HeBA2)
ResNet-50	23.5	19.5	(25.6, 20.5, 17.3)	(26.3, 22.9, 18.0)	(26.9, 21.4, 19.6)	(28.2, 24.4, 19.7)	(27.8, 24.1, 20.0)	(18.2, **15.8**)
VGG-16	22.0	18.7	(23.2, 19.6, 16.9)	(24.7, 20.5, 17.4)	(23.1, 19.2, 17.1)	(25.4, 21.0, 18.2)	(25.5, 22.7, 19.2)	(16.5, **14.2**)
EfficientNet	18.2	17.2	(22.5, 19.0, 15.7)	(22.2, 18.2, 15.4)	(21.3, 18.6, 14.9)	(24.6, 21.2, 18.7)	(22.8, 18.5, 15.9)	(16.1, **12.7**)
MobileNet-V3	15.7	15.1	(18.2, 15.6, 13.0)	(16.3, 13.7, 11.5)	(17.4, 15.2, 12.4)	(18.8, 15.7, 13.4)	(17.2, 14.9, 12.0)	(11.6, **9.5**)

TABLE X

THE MIOU (%) WHEN $\xi = 10$. BOLD NUMBER MARKS THE LOWEST MIOU IN EACH ROW. UNDERLINED NUMBER SHOWS THE RUNNER UP PERFORMANCE.

	GAP	TDA	(UAT, HoBA1, HoBA2)	(DF, HoBA1, HoBA2)	(COS, HoBA1, HoBA2)	(NAG, HoBA1, HoBA2)	(TRM, HoBA1, HoBA2)	(HeBA1, HeBA2)
FCN-Alex	10.7	11.7	(14.5, 12.6, 10.5)	(14.2, 11.9, 10.2)	(15.0, 13.1, 11.3)	(14.7, 12.9, 11.1)	(15.1, 13.7, 11.2)	(11.0, **9.4**)
FCN-8s-VGG	25.6	23.9	(28.5, 24.6, 21.0)	(27.8, 24.2, 20.8)	(28.2, 25.1, 22.6)	(30.5, 25.8, 23.4)	(29.2, 25.3, 22.6)	(23.3, **20.6**)
DL-VGG	36.2	37.6	(40.6, 37.2, 33.0)	(38.5, 35.8, 32.2)	(38.8, 35.2, 32.8)	(41.2, 38.2, 34.5)	(40.2, 37.8, 34.0)	(34.2, **31.1**)
DL-RN101	29.0	27.5	(33.5, 30.2, 28.5)	(32.1, 29.4, 26.2)	(30.7, 27.3, 25.4)	(34.0, 31.8, 28.1)	(32.7, 28.9, 26.9)	(27.1, **23.9**)

diversity of UAPs is still a key factor contributing to the higher attack effectiveness of the boosting attack framework.

APPENDIX F
ATTACK EFFECTIVENESS IN SEMANTIC SEGMENTATION (Q13)

In this section, we answer Q13 by examining the attack effectiveness of the proposed boosting attack methods in the semantic segmentation task.

We follow [29] to extend our methods and the single-UAP methods such that they can be applied to attack a victim segmentation model. Specifically, for each of these methods, we use its attack function $L(x_i, \delta_j)$ to measure the similarity between the predictions of each pixel made by the victim segmentation model before and after the perturbation δ_j is added to the target images x_i. For the baseline methods GAP and TDA, they can be directly applied to attack segmentation models without any extension [28], [29].

Following the experimental settings in [29], we use the training dataset of Cityscapes [61] to train the attack model of each attack method and use the validation dataset of Cityscapes to evaluate the attack effectiveness of different attack methods. We choose FCN-Alex [62], FCN-8s-VGG [62], DL-VGG [63] and DL-RN101 [63] as the victim segmentation models, which are pretrained on the training dataset of Cityscapes. We use the *mean intersection over union (mIoU)* [29] of a victim segmentation model achieves on the testing dataset as the metric to evaluate the attack effectiveness of an attack method. A lower mIOU indicates worse predictive performance of the victim segmentation model, thus implying better attack effectiveness of that attack method. We report mIOU in percentage by default. The other experimental settings are the same as in Section VI-B and we use $\xi = 10$ for all the attack methods.

Table X reports the results of mIOU of all the attack methods. We can see that when the proposed boosting attack framework is applied, the mIOU of the single-UAP methods is significantly decreased. In addition, the mIOU of UAT-HoBA2, DF-HoBA2 and HeBA2 is almost always lower than that of GAP and TDA. These results demonstrate the great performance of the proposed boosting attack methods, indicating that the boosting attack framework is also effective in improving the attack effectiveness of the single-UAP methods in the semantic segmentation task. Moreover, we can also see that the lowest mIOU is consistently achieved by HeBA2 as shown by the bold numbers in each row of Table X. Such results highlight the importance of improving the diversity of UAPs, which is critical when attacking segmentation models using the proposed boosting attack framework to achieve high attack effectiveness.

APPENDIX G
POTENTIAL LIMITATIONS (Q14)

Here, we answer Q14 by discussing the potential limitations of the proposed boosting attack framework.

The boosting attack framework is designed to be generally applied to existing single-UAP methods, in order to improve their attack performance. However, existing single-UAP methods primarily perform in the digital world and how to effectively apply them to the physical world has not been systematically studied in the literature [6], [13]–[17], [21]–[25]. Thus, it remains to be explored to apply the boosting attack methods in the physical world.

Nevertheless, the key idea of the boosting attack framework that utilizes multiple diversified perturbations to significantly improve attack effectiveness has good potential to be extended for attacks in the physical world. For example, some recent works [64], [65] have proposed *universal adversarial patch* – a small, localized image that, when physically placed anywhere in an input, can mislead a victim DNN into making a wrong prediction. Such a patch can be directly applied in the physical world and has shown good attack performance in a variety of applications such as autonomous driving [66], face recognition [65], automatic check-out [64], etc. However, the number of inputs that can be successfully attacked by a single universal adversarial patch may be limited. Therefore, the key idea of our boosting attack framework may be applied to generate multiple diversified universal adversarial patches, such that the performance of the patch-based attacks in the physical world can be further improved. We will investigate this problem in the near future.

Another potential limitation of our boosting attack framework is that UAPs generated for one task (e.g., classification) are not very effective in launching attacks for another task (e.g., object detection) [28], [67]. While our framework has demonstrated great transfer attack effectiveness within the same task as shown by the experimental results in Table III, its attack effectiveness across different tasks is relatively limited. Nevertheless, we emphasize that the boosting attack framework poses a new and serious security threat to DNNs by revealing a previously unexplored approach that can significantly enhance the effectiveness of UAP-based attacks against DNNs. This also offers practical value for more comprehensively evaluating the robustness of DNNs, thereby motivating future research into building more secure and robust DNN models.