8. Appendix

8.1. Proof of Linear Translation Invariance of the Softmax Function

We now prove in detail that the softmax function is invariant under linear translation. Specifically, we will show that if the input vector to the softmax function is linearly shifted by a constant, the output remains unchanged.

Let $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$ be a real-valued vector. The softmax function applied to $\mathbf{y}$ produces a vector $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ where each component Y_i is given by:

$$Y_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{31}$$

Let $c \in \mathbb{R}$ be a constant, and consider the translated vector $\mathbf{y}' = (y_1 + c, y_2 + c, \ldots, y_n + c)$. We aim to prove that the softmax of the translated vector $\mathbf{y}'$ is equal to the softmax of the original vector $\mathbf{y}$. For the translated vector $\mathbf{y}'$, the softmax output is:

$$Y_i' = \frac{e^{y_i + c}}{\sum_{j=1}^{n} e^{y_j + c}} \tag{32}$$

We can simplify the expression by factoring out e^c from both the numerator and denominator:

$$Y_i' = \frac{e^{y_i} e^{c}}{\sum_{j=1}^{n} e^{y_j} e^{c}} \tag{33}$$

Since e^c is a common factor in both the numerator and the denominator, it cancels out:

$$Y_i' = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} = Y_i \tag{34}$$

This expression is exactly the definition of s_i, the softmax component of the original vector $\mathbf{z}$. Since $Y_i' = Y_i$ for all i, we conclude that the softmax function is invariant under translation. That is, if we translate the input vector $\mathbf{y}$ by adding a constant c to each element, the softmax output remains unchanged, which is:

$$\text{softmax}(\mathbf{y}' = \mathbf{y} + c) = \text{softmax}(\mathbf{y}) \tag{35}$$

With such *Linear Translation Invariance of the Softmax Function*, we can maximize the L_2 norm of logits output $||\mathbf{y}||_2$ and its variance $\sigma(\mathbf{y})$ simultaneously to place the fingerprint samples close to the *steep decision boundary* for optimizing the fingerprint samples' sensitivity with respect to model tampering.

8.2. Detaled explanation on optimization efficiency

Different from [13], we do not directly optimize Eq.5 and Eq.13 as (1) Using Eq.5 and Eq.13 as the loss function requires computing the second derivative during backpropagation optimization because calculating the loss function Eq.5 and Eq.13 involves first-order derivatives. This significantly reduces optimization efficiency. (2) In existing deep learning frameworks (e.g., PyTorch and TensorFlow), directly optimizing Eq.13 would cause the program to work serially, preventing the parallel optimization of a batch of fingerprint samples x. This is because, in PyTorch and TensorFlow, logits vector of a batch sample X, $f(W, X) = [\mathbf{y}_{x_1}, \mathbf{y}_{x_2}, \cdots, \mathbf{y}_{x_n}]$, cannot directly compute the gradient with respect to parameter W; instead, operation $f(W, X).sum(\,)$ is needed. Thus, if Eq.13 is used as the loss function, the gradients of different fingerprint samples in a batch would be the same in the input space, which is clearly incorrect. This would further reduce optimization efficiency.

8.3. Results on ImageNet

Tab.3 summarizes the tampering detection results on ImageNet. Note that due to bit-flipping causing the model to fail in properly classifying any inputs, the relationship between the detection rate and K is no longer stable for both IBSF and SDBF.

8.4. Results on Smoothed Model

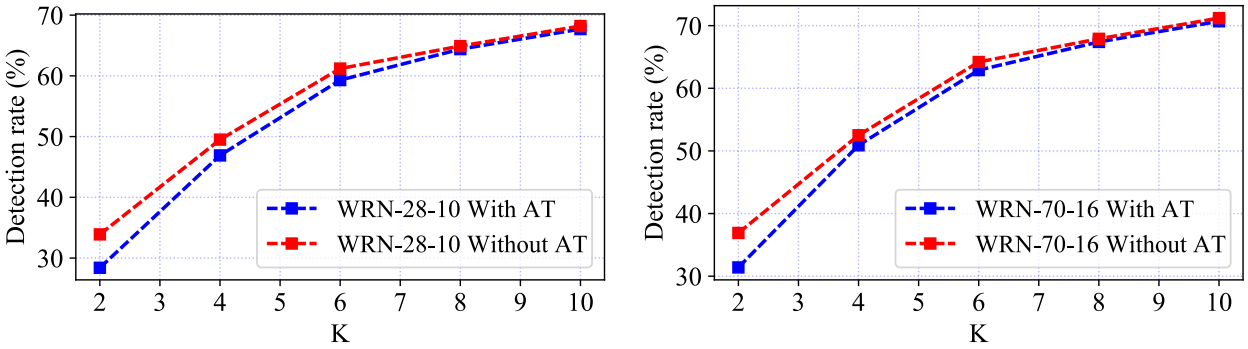
Since our SDBF leverages the steep decision boundary of a classification model, a natural question is whether our SDBF is applicable to the smoothed model. To answer this, we use two models smoothed with an SOTA *adversarial training(AT)* method pre-trained by [32] on CIFAR10, which is WRN-28-10 and WRN-70-16, to explore whether our SDBF is applicable to such smoothed models. Tab.4 summarizes the information on these two models, including clean accuracy, robust accuracy under auto-attack [5], and the total number of parameters.

We evaluate the impact of *adversarial training(AT)* on both detection performance and optimization efficiency of SBDF. The results are shown in Fig.8 and Fig.9. We find that although AT slightly increased the optimization time cost of SDBF and slightly reduced its performance when $K = 2$, SDBF remains efficient and effective when K is larger. This is an interesting yet reasonable phenomenon.

An intuitive explanation is that **the decision boundary between any two classes and the decision boundary among $K (K \geqslant 3)$ classes are two distinct regions** based on Def.4.1. While adversarial training smooths the decision boundary between any two classes, including the true class and adversarial target class (non-true class) of $\mathbf{x}$ (meaning $K = 2$ in this paper), it **does not explicitly enforce the smoothing of the intersecting decision boundary of K $(K \geqslant 3)$ classes**, making the smoothness of K $(K \geqslant 3)$-class decision boundary might not be as pronounced as that of any two classes ($K = 2$). Notably, this characteristic of

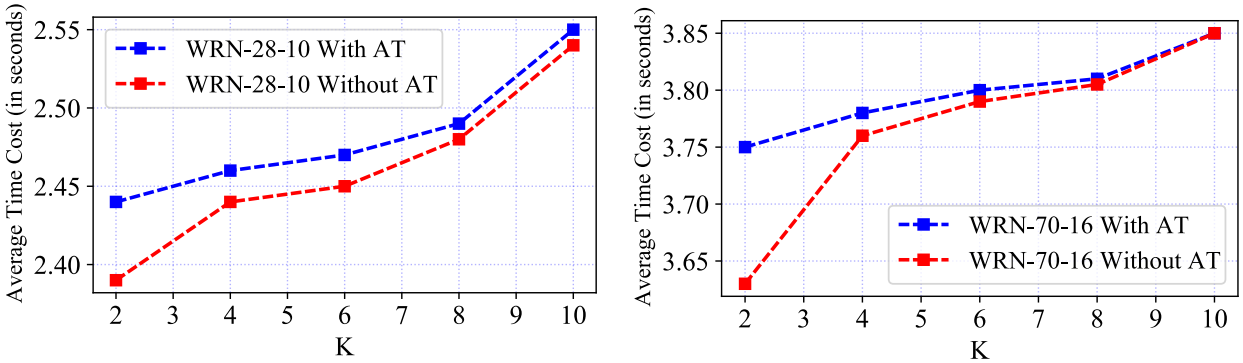
Table 3. Tamper detection rate (%) with $M = 1$ probe on ImageNet. For fine-tuning(FT), "Last" specifies only fine-tuning the last layer, with numbers indicating the learning rate exponent (e.g., 10^{-5}). The 'On-Learning' refers to online learning.

Method		FT_{Last}-5	Backdoor	T-Attack	Unlearning	Quantization	Different Arch.	Pruning	On-Learning	Bit-fliping
SSF		35.4	67.1	50.2	43.1	81.2	82.6	72.2	56.8	91.2
MiSentry		84.9	89.8	89.5	86.6	92.5	93.8	91.3	88.3	92.3
PublicCheck		42.5	82.4	65.2	59.4	88.3	84.5	79.6	55.9	92.8
IBSF	$K = 2$	56.3	84.9	75.6	68.8	90.4	87.1	85.5	66.1	91.1
	$K = 4$	68.9	88.2	83.3	71.3	89.1	89.3	91.5	78.8	90.4
	$K = 6$	79.4	90.3	89.9	79.5	90.4	92.2	91.9	86.9	92.3
	$K = 8$	86.6	90.9	90.4	86.8	91.1	93.6	92.4	89.3	92.1
	$K = 10$	88.7	91.2	91.5	89.2	91.3	94.8	93.3	90.5	93.9
SDBF(ours)	$K = 2$	62.1	88.7	81.5	76.4	90.5	91.2	89.4	73.9	92.8
	$K = 4$	83.2	90.4	87.1	85.2	92.6	92.8	92.8	85.1	91.5
	$K = 6$	87.5	91.6	89.6	90.3	92.8	94.1	94.2	89.9	91.9
	$K = 8$	89.5	91.9	90.5	91.2	93.5	94.4	94.3	91.8	93.7
	$K = 10$	91.4	93.3	92.7	93.9	94.7	96.1	95.4	93.5	94.2



(a) WRN-28-10 (b) WRN-70-16

Figure 8. (a) Tampering detection rates of WRN-28-10 on CIFAR10 using a single fingerprint sample for K ranging from 2 to 10 to detect fine-tuning the last layer with 10^{-5} learning rate. (b) Tampering detection rates of WRN-70-16 on CIFAR10 using a single fingerprint sample for K ranging from 2 to 10 to detect fine-tuning the last layer with 10^{-5} learning rate.



(a) WRN-28-10 (b) WRN-70-16

Figure 9. (a) Average time cost (in seconds) of WRN-28-10 on CIFAR10 using a single fingerprint sample for K ranging from 2 to 10 to detect fine-tuning the last layer with 10^{-5} learning rate. (b) Average time cost (in seconds) of WRN-70-16 on CIFAR10 using a single fingerprint sample for K ranging from 2 to 10 to detect fine-tuning the last layer with 10^{-5} learning rate.

Table 4. Clean accuracy, robust accuracy, and number of parameters on CIFAR10.

Model	Clean Acc	Robust Acc	# of params
WRN-28-10	92.44%	67.31%	20M
WRN-70-16	93.25%	70.69%	50M

adversarial training **does not negatively impact its robustness** to adversarial examples since each adversarial example has a true label and a target label (non-true label). However, this creates an opportunity for SDBF to be applicable, es-

pecially when $K \geqslant 3$, where it remains both effective and efficient.

In this paper, we have preliminarily investigated whether SDBF can be applied to smoothed models. The question of how adversarial training precisely smooths the decision boundaries of classes is left for future work. Overall, SDBF is applicable to both models trained with standard methods and those smoothed using current state-of-the-art adversarial training techniques.