MetaFinger: Fingerprinting the Deep Neural Networks with Meta-training

Kang Yang[1,2] , **Run Wang**[1,2*] , **Lina Wang**[1,2,3]

[1]School of Cyber Science and Engineering, Wuhan University, China

[2]Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, China

[3]Zhengzhou Xinda Institute of Advanced Technology

Abstract

As deep neural networks (DNNs) play a critical role in various fields, the models themselves hence are becoming an important asset that needs to be protected. To achieve this, various neural network fingerprint methods have been proposed. However, existing fingerprint methods fingerprint the decision boundary by adversarial examples, which is not robust to model modification and adversarial defenses. To fill this gap, we propose a robust fingerprint method **MetaFinger**, which fingerprints the inner decision area of the model by meta-training, rather than the decision boundary. Specifically, we first generate many shadow models with DNN augmentation as meta-data. Then we optimize some images by meta-training to ensure that only models derived from the protected model can recognize them. To demonstrate the robustness of our fingerprint approach, we evaluate our method against two types of attacks including input modification and model modification. Experiments show that our method achieves 99.34% and 97.69% query accuracy on average, surpassing existing methods over 30%, 25% on CIFAR-10 and Tiny-ImageNet, respectively. Our code is available at https://github.com/kangyangWHU/MetaFinger/

1 Introduction

In recent years, deep neural networks (DNNs) have achieved state-of-the-art performance in a wide range of fields, including computer vision, speech recognition, autonomous driving, etc. Training such a model is a non-trivial task that requires tedious data collection, time-consuming model training, and careful model testing. Hence the models are becoming the business assets and need to be guarded against stealing, and illegal redistribution.

To protect the intellectual property (IP) of the owner's model (the source model), two groups of works have been proposed: model watermark and model fingerprint. Model watermark methods either embed bit messages into the model weights or force the model to remember a query set that is used to verify the identity of suspicious models. Whichever model watermark techniques the owner applies, it inevitably sacrifices the utility of the model. Recent work [Lukas *et al.*, 2021] also points that existing model watermark approaches are not robust against removal attacks. Model fingerprint methods [Cao *et al.*, 2021; Lukas *et al.*, 2019] extract the fingerprint from trained models without modifying the model at all, which shows great potential to protect the model. They first construct a query set from the source model which can prove the identity of the source model, then they query the suspicious model with the query set. If the outputs match the source model with a high rate, they judge that the suspicious model is a copy from the source model.

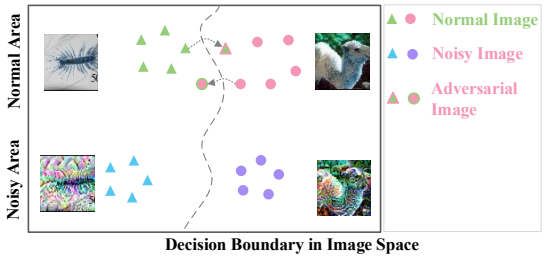


Figure 1: The intuition of our approach.

However, existing fingerprint methods fingerprint the decision boundary by adversarial examples [Szegedy *et al.*, 2013], which suffer from the following problems. First, the decision boundary is likely to move under model modification, which makes the fingerprint methods less reliable. Second, adversarial examples can be defeated by adversarial defenses such as input modification [Xu *et al.*, 2017] and adversarial training [Zhang *et al.*, 2019], hence it may fail to identify the real stolen models.

To solve those problems, in this paper, we present a robust neural fingerprint method **MetaFinger** which fingerprints the inner decision area where images in this area can only be recognized by models derived from the source model. The intuition of our approach is illustrated in Fig. 1. According to Nguyen et al. [2015], DNNs also classify unrecognizable images with high confidence. Inspired by this, we assume there are two decision areas in the image space for a model: normal area that is full of normal images and noisy area that consists of noisy images. For natural images belonging to the

*Run Wang is the corresponding author (wangrun@whu.edu.cn)

normal area, they can be successfully classified by all well-trained models, so they cannot be applied to fingerprint the model. But for noisy images, even well-performed models have large divergences. Only models with similar decision boundary predict those images with the same labels because their decision boundary has a large overlap. Therefore, we can adopt noisy images to fingerprint the inner decision area of the source model.

Specifically, our approach includes two stages: meta-data generation and meta-training. In the meta-data generation stage, we first train a source model and some negative models from scratch. Then inspired by data augmentation, we apply DNN augmentation operation on the models to construct a positive model pool and a negative model pool as meta-data. Then in the meta-training stage, we optimize the images with meta-training guided by triple loss to ensure the images can only be recognized by models that stem from the source model. Finally, we screen out the successful images to construct the query set.

The main contributions are summarized as follows:

- We propose MetaFinger, a robust DNN fingerprint approach for protecting the IP of the owner's model. It optimizes some images to fingerprint the inner decision area of the source model by meta-training.
- To reduce the overhead of meta-training, we propose DNN augmentation that augments the DNN with noise and fine-tunes it. Experiments show that our DNN augmentation strategy significantly reduces the number of required models and improves the query accuracy.
- Experiments demonstrate that MetaFinger is robust against two types of removal attacks including input modification, model modification. Compared with previous methods, our approach achieves 99.34% and 97.69% query accuracy on average, exceeding current methods over 30%, 25% on CIFAR-10, and Tiny-ImageNet dataset, respectively.

2 Related Work

2.1 Model Watermark

Existing model watermark methods can be grouped into two categories: parameter based and query set based. Parameter based watermark methods [Uchida *et al.*, 2017; Darvish Rouhani *et al.*, 2019; Wang and Kerschbaum, 2021] embed messages into the model parameters by adding new regularization terms to the task loss, then extract those messages during white-box verification. Uchida et al. [2017] first embed bit messages into the weights during the training process by combining task loss and embedding loss. To keep the distribution of model parameters undetectable, Wang et al. [2021] propose RIGA, which leverages the idea of generative adversarial networks to embed the messages.

Query set based watermark methods first build a query set from the source model, then they query the suspicious model in black-box setting. If the suspicious models predict the same labels for most of the query samples (greater than a threshold), they consider the model is illegally copied. Many works [Adi *et al.*, 2018; Li *et al.*, 2019; Zhang *et al.*, 2018; Jia *et al.*, 2021; Szyller *et al.*, 2021] utilize backdoor attacks to construct the query set. They implant triggers or samples into the model during training or fine-tuning as done in backdoor attacks. Once embedded with the backdoor, the source model will output specific labels for the query set. Some other works [Le Merrer *et al.*, 2020; Chen *et al.*, 2019] leverage adversarial examples to construct the query set.

2.2 Model Fingerprint

Instead of inserting the watermark into the source model, model fingerprint methods directly extract a fingerprint from the trained source model without affecting the model. Cao et al. [2021] use adversarial examples from the decision boundary as query set, and Lukas et al. [2019] use conferable adversarial examples that only transfer to its pirated model as the query set. If the suspect model predicts the same labels for most queries, they consider the suspect model is potentially pirated from the source model. Both methods leverage the adversarial examples to fingerprint the model, which suffers from adversarial defenses.

2.3 Meta-Training

Unlike traditional machine learning tasks that learn from data samples like texts and images, meta-training learns from neural networks. It aims to learn whether a model has some property, such as whether a model has a dropout layer. Ateniese et al. [2015] first utilize the representation of the classifier to train a meta-classifier that discovers some statistical information about the training set. Oh et al. [2019] feed query inputs into the shadow models and feed its outputs into a meta-classifier, then it simultaneously optimizes both the query inputs and the meta-classifier. Similar to Oh [2019], Xu et al. [2021] propose MNTD to detect whether the model is embedded with a backdoor. Our approach does not train a meta-classifier, because we aim to construct a query set rather than simply decide whether the model is pirated or not.

3 Problem Definition

3.1 Threat Model

We consider two parties, a model owner who trains the source model, and an attacker who has stolen the source model through illegal copy.

The goal of the model owner is to identify the stolen models that are remotely deployed by the attacker. The model owner has (1) white-box access to the source model, (2) black-box access to the model to be verified.

The attacker's goal is not verified as the pirated version of the source model while keeping a similar performance as the source model. To achieve this, the attacker can attempt to modify the query samples (input modification) or change the model (model modification). We assume the attacker has some validation data from the same distribution of training data to fine-tune the model.

3.2 Guarding IP via Fingerprint DNNs

Protecting the IP of DNNs via fingerprinting methods consists of two stages: fingerprint extraction and ownership verification. We describe those two stages as follows.

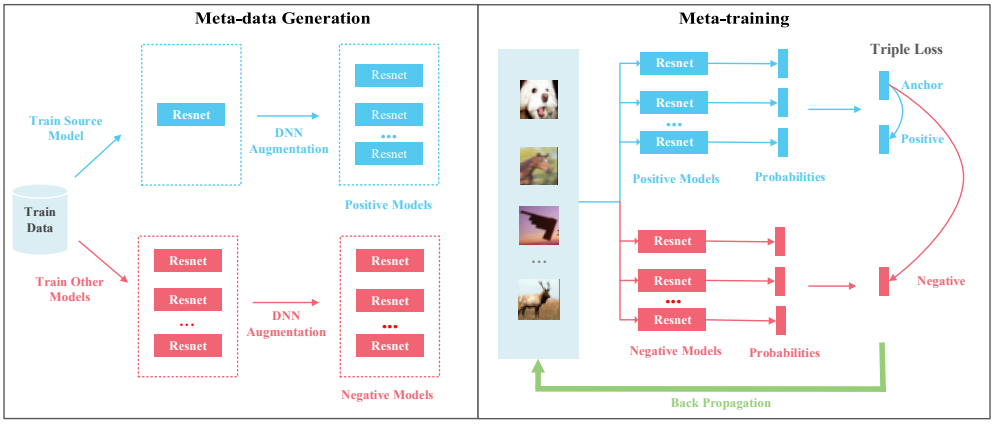


Figure 2: The illustration of our framework.

Fingerprint Extraction. Given white-box access to a source model M, the Extract function outputs a fingerprint as $FP_M = Extract(M)$.

Ownership Verification. Given black-box access to a suspicious model V, $Verify(V, FP_M)$ predict either 0 or 1, where 1 denotes the suspicious model is an illegal copy from the source model M.

3.3 Requirements for DNN Fingerprint

We aim to design a DNN fingerprint algorithm that satisfies the following properties.

- **Fidelity.** The extraction of the fingerprint should not affect the performance of the source model.
- **Robustness.** The algorithm should be robust against the attacks such as input modification or model modification, which aims to escape the ownership verification.
- **Reliability.** The algorithm should identify the illegal coped models with a high probability.
- **Integrity.** The algorithm should not charge honest parties with similar models for theft.
- **Efficiency.** The algorithm should not add a large overhead on the process of fingerprint extraction and verification.

4 Method

As shown in Fig. 2, MetaFinger contains two stages: meta-data generation and meta-training. In meta-data generation stage, it trains and augments several shadow models as meta-data. In the meta-training stage, MetaFinger optimizes some images guided by triple loss to ensure that the images can only be recognized by models stem from the source model.

4.1 Meta-Data Generation

In meta-data generation step, we first train some models from scratch as negative models. Then inspired by data augmentation, we apply augmentation on models to establish a positive model pool and a negative model pool as meta-data. In this paper, we adopt weights noise strategy that first adds Gaussian noise on the weights layer by layer then fine-tunes the model to achieve high accuracy as:

$$param+ = GaussianNoise(0, std(param)/\alpha) \quad (1)$$

where $param$ denotes the parameters of one layer, and $std(\cdot)$ stands for the variance function. The α controls the strength of perturbation, and small α encourages adding large noise.

4.2 Meta-Training

In this stage, we craft a query set via meta-training. We first split the meta-data into train data $\mathcal{D}_{train}$ and validation data $\mathcal{D}_{val}$. The train data are used to optimize the images and the validation data are used to evaluate the performance and screen out the final query set. Unlike traditional meta-training that trains a meta-classifier, we aim to construct a query set that can be applied to verify the ownership of the model.

Our training process is inspired by metric learning with triple loss that aims to reduce inner-class distance and enlarge intra-class distance. But there is a distinct difference that we only reduce the inner-class distance for positive class, not for all classes. The reason is that we only require positive models predict the same label that is different from the label predicted by negative models. We do not force all negative models to predict the same label as this may make it hard to optimize. To achieve this, we adopt triple loss as the loss function and only choose the anchor point from the positive class.

Before starting the meta-training process, we need to initialize the images X. One way is initialized with noise, and the other way is starting from random train data. We find that noisy images are hard to optimize because they are always stuck to fixed classes. So we randomly choose some train data as the starting point of images X. After giving necessary inputs, as shown in Algorithm 1, the meta-training algorithm first trains one epoch (line 6 $\sim$ 14), then evaluates its performance to avoid over-fitting (line 15 $\sim$ 20), finally screens out successful images as the query set (line 22).

Train One Epoch (line 6 $\sim$ 14). For each epoch, we take the model from the positive models in turn as the anchor model p_{anchor}, then randomly select k positive models $p_i, i = \{1, 2, ..., k\}$ and k negative models $n_i, i = \{1, 2, ..., k\}$ from the corresponding model pool. We feed the trainable images X into those models and obtain their predicted probabilities. Then we compute their triple loss as:

Algorithm 1 Meta-training

Input: Meta-data $\mathcal{D}$, Images X, Learning Rate η, Train Epoch N_{epoch}, Loss Control λ, Number of Sample k.
Output: Query set $Q(X', Y')$

1: $X_{best}, ACC_{best} = init\_param()$
2: $\mathcal{D}_{train}, \mathcal{D}_{val} = train\_val\_split(\mathcal{D})$
3: $\mathcal{P}_{train}, \mathcal{N}_{train} = postivate\_negative\_split(\mathcal{D}_{train})$
4: $\mathcal{P}_{val}, \mathcal{N}_{val} = postivate\_negative\_split(\mathcal{D}_{val})$
5: **for** $n = 1$ to N_{epoch} **do**
6: **for** p_{anchor} in $\mathcal{P}_{train}$ **do**
7: $p_1, p_2, ..., p_k = random\_select(\mathcal{P}_{train}, k)$
8: $n_1, n_2, ..., n_k = random\_select(\mathcal{N}_{train}, k)$
9: $L_{pos} = \frac{1}{k}\sum_{i=1}^{k} KL(p_{anchor}(X), p_i(X))$
10: $L_{neg} = \frac{1}{k}\sum_{i=1}^{k} KL(p_{anchor}(X), n_i(X))$
11: $Loss = L_{pos} - \lambda L_{neg}$
12: $X = X - \eta \nabla Loss$
13: $X = clip(X, 0, 1)$
14: **end for**
15: $Y = get\_label(X, \mathcal{P}_{val})$
16: $ACC_{neg} = get\_mean\_acc(\mathcal{N}_{val}, X, Y)$
17: **if** $ACC_{neg} < ACC_{lowest}$ **then**
18: $ACC_{lowest} = ACC_{neg}$
19: $X_{best} = X$
20: **end if**
21: **end for**
22: $Q = screen(X_{best})$
23: **return** Q

$$
\begin{aligned}
Loss = & \frac{1}{k}\sum_{i=1}^{k} KL(p_{anchor}(X), p_i(X)) \\
& - \lambda \frac{1}{k}\sum_{i=1}^{k} KL(p_{anchor}(X), n_i(X))
\end{aligned} \tag{2}
$$

Where KL denotes Kullback-Leibler divergence that is utilized to measure the distance of two probability distributions. λ is applied to control the intra-class distance and larger λ pays more attention to reducing intra-class distance. k is adopted to smooth the training process. The first term of loss function minimizes the inner-class distance of positive models to encourage them to output the same results. The second term maximizes the intra-class distance that forces negative models to output different results as positive models. Then we can optimize the X according to the gradient of loss.

Evaluate the Performance (line 15 $\sim$ 20). After one epoch, we evaluate the performance of X and save the best X as X_{best} using validation data $\mathcal{D}_{val}$. Specifically, we regard the labels predicted by most positive models as ground-truth labels Y. Then we calculate the mean accuracy of positive models ACC_{pos} and negative models ACC_{neg}. Here, we only consider ACC_{neg} because we find the ACC_{pos} always close to 1. We save the X_{best} that has the lowest ACC_{neg}.

Screen Query Set (line 22). After training, we screen out the successful images from X_{best} with validation data $\mathcal{D}_{val}$ to build the query set. For $x \in X_{best}$ and its corresponding label $y \in Y$, if all positive models predict it as y, as well as no negative models predict it as y, we add it to the query set.

5 Experiments

To demonstrate the effectiveness and robustness of MetaFinger, we run experiments against two types of removal attacks including input modification and model modification on CIFAR-10 and Tiny-ImageNet benchmark datasets. We also compare our method with two advanced model watermark methods Adi [Adi *et al.*, 2018], Jia [Jia *et al.*, 2021], and one DNN fingerprint method IPGuard [Cao *et al.*, 2021]. We implement Adi in two different ways: training from scratch and fine-tuning the pretrained model, which are referred to Adi(Scratch) and Adi(Pretrained), respectively. We only implement Jia by fine-tuning the pretrained models.

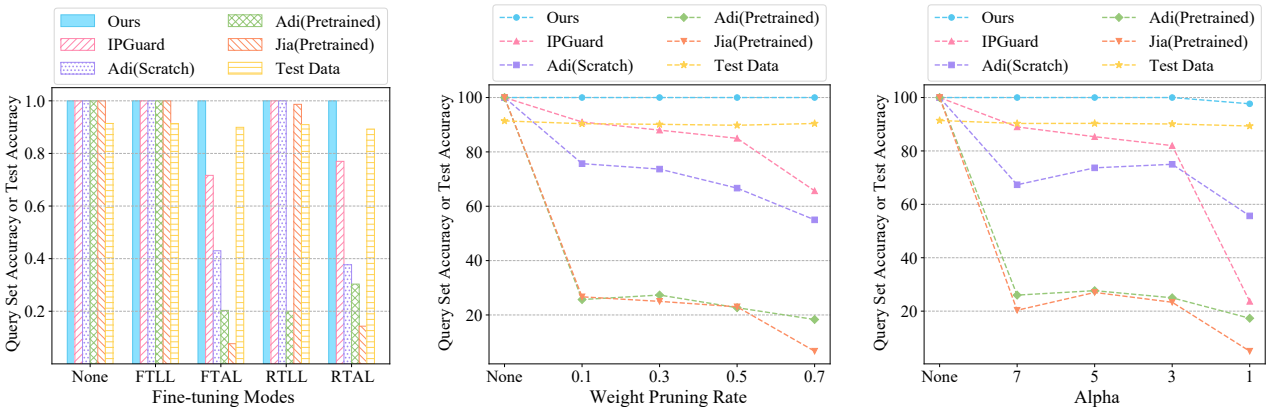
5.1 Removal Attacks

Input Modification. Input modification applies preprocessing operation on the inputs before feeding them into the model for adversarial defense. It can also be adopted by the attacker to remove the noise in the query data. We apply three different preprocessing image transformations as follows.

- Random Resize and Padding (RP) [Xie *et al.*, 2017]. RP first resizes the image to a random size, then padding it to a fixed size.
- Input Noising [Dziugaite *et al.*, 2016]. Input noising adds some noises to the images which may interfere with the adversarial noise. We add Gaussian noise and universal noise in the experiments.
- Input Smoothing [Xu *et al.*, 2017]. Input smoothing convolves the image with some kernel, which makes the image blurry. We adopt mean, median, and Gaussian kernel in the experiments.

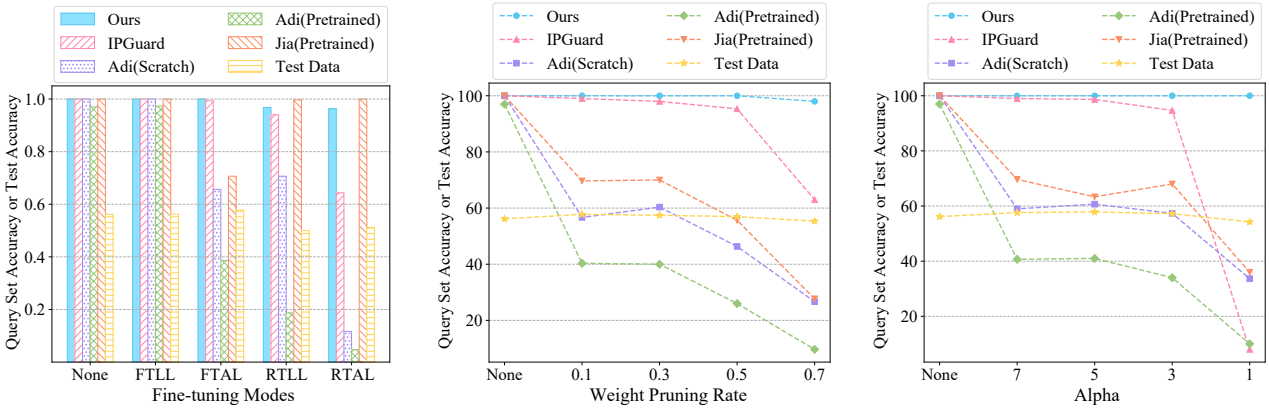
Model Modification. Model modification alters the source model's parameters to remove the watermark or fingerprint. We apply fine-tuning, weight pruning, and weight noising to modify the model.

- Fine-tuning. we fine-tune the model with validation data in the following four modes as done in [Adi *et al.*, 2018].
  - Fine-Tune Last Layer (FTLL): Fine-tune the last layer in the model and freeze all other layers.
  - Fine-Tune All Layers (FTAL): Fine-tune all layers in the model.
  - Re-Train Last Layer (RTLL): Randomly initialize the weights of the last layer and fine-tune them.
  - Re-Train All Layers (RTAL): Randomly initialize the weights of the last layer and fine-tune all layers.
- Weight pruning. Weight pruning has long been used to compress the model by removing redundant weights. We apply the same pruning algorithm in [Han *et al.*, 2015], which prunes parameters with small absolute values. To restore the accuracy, we assume the attacker fine-tunes the pruned models.
- Weight Noising. Weight noising adds noise to the weights of the model as in Eq. (1), then fine-tunes it.



(a) Fine-tuning (b) Weight Pruning (b) Weight Noising

Figure 3: The query set accuracy under (a) Fine-tuning, (b) Weight Pruning, and (c) Weight Noising on CIFAR-10 dataset.



(a) Fine-tuning (b) Weight Pruning (b) Weight Noising

Figure 4: The query set accuracy under (a) Fine-tuning, (b) Weight Pruning, and (c) Weight Noising on Tiny-ImageNet dataset.

5.2 Robustness against Model Modification

To escape the identification of IP protection methods, the attacker may alter the weights of the stolen model with fine-tuning, weight pruning, and weight noising.

Fine-tuning. Fig. 3 (a) and Fig. 4 (a) show the results of fine-tuning on CIFAR-10 and Tiny-ImageNet. We observe that almost all methods still achieve 100% query accuracy when only modifying the weights of the last layer except Adi(Pretrained) on CIFAR-10. But when fine-tuning all the weights, the query accuracy of backdoor based methods sees a large decline, especially those pretrained methods. Differing from watermarking methods, our method can achieve over 95% query accuracy under all fine-tuning operations.

Weight Pruning. We prune $p\%$ (from 10% to 70%) of parameters with the smallest absolute value at each layer by setting them to zero, then fine-tune the model. As shown in Fig. 3 (b) and Fig. 4 (b), the query accuracy of all evaluated methods except our approach gradually declines as the pruning ratio increases. But the test accuracy is still able to recover to the original accuracy even when removing 70% weights. Only our approach achieves over 95% query accuracy under all pruning rates.

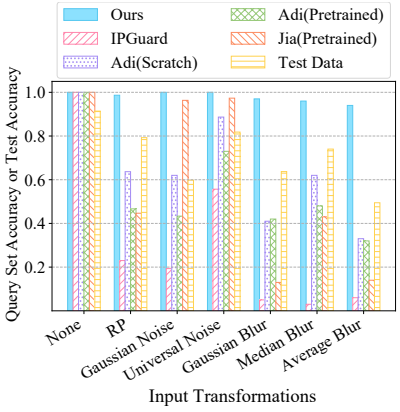


Figure 5: The query set accuracy under input modification.

Weight Noising. We perturb the model weights with Eq. (1) and set α to different values, then fine-tune the model. Fig. 3 (c) and Fig. 4 (c) show that as the strength of added noises increases (small α), the query accuracy of all other methods begins to drop and eventually below 60%. In contrast, our

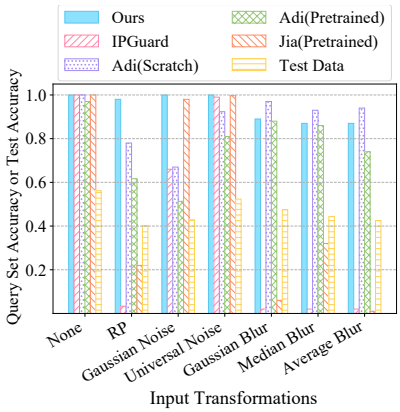


Figure 6: The query set accuracy under input modification.

approach only slightly drops to around 95% under the largest weight noise.

5.3 Robustness against Input Modification

Input modification preprocesses the inputs with image transformations to defense adversarial examples, and it can also be utilized to hinder the verification of DNN fingerprint methods. Fig. 5 and 6 show the results of all evaluated methods against six input modification methods on CIFAR-10 and Tiny-ImageNet. We observe that the accuracy of all methods has degraded to some extent under input transformations. It is worth noting that although the test accuracy drops a lot, our method still achieves over 80% accuracy regardless of the input transformations. We conjecture that our method adds many robust perturbations which can be recognized by the classifier. Among all the methods, IPGuard performs the worst because their query set is constructed with adversarial examples which are not robust against image transformations. Even the backdoor based methods do not add perturbations on the inputs, the image transformations also significantly reduce their query accuracy.

5.4 Integrity

Instead of identifying stolen models with high query accuracy, DNN fingerprint and watermark methods should not judge clean models as pirated models. Therefore, the query accuracy of clean models should be as low as possible. Tab 1 gives the query accuracy on four different model architectures on CIFAR-10. From this table we can observe that our approach does not lead to high accuracy for clean models with different network architectures, even our approach only trains on Resnet architecture. IPGuard has the largest query accuracy on Resnet because their query data are full of adversarial examples which easily transfer to similar network architectures. Jia(Pretrained) yields the smallest query accuracy on clean models because their query data consist of normal images with patch, which can only be predicted with the specific labels by backdoored models.

5.5 Efficiency

A good IP protection method should also not put a large overhead on the model owner. For our approach, the overhead

Methods	Resnet	VGG	InceptionV3	DenseNet
Ours	4.00%	4.67%	12.33%	4.00%
IPGuard	13.00%	0.67%	0.67%	5.00%
Adi(Scratch)	12.67%	10.67%	11.00%	9.67%
Adi(Pretrained)	12.33%	12.00%	10.33%	10.67%
Jia(Pretrained)	0.33%	0.00%	0.00%	0.33%

Table 1: The query accuracy of all methods on clean models.

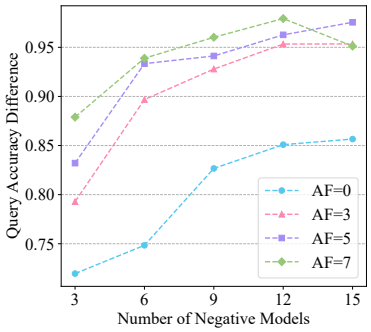


Figure 7: The average query accuracy difference between all positive models and all negative models on CIFAR-10. AF is short for augmentation factor, and "AF=3" denotes that augments three models for one negative model.

lies in training negative models. To solve this problem, we adopt DNN augmentation to increase the number of negative models. Therefore, in this section, we investigate the impact of negative models and DNN augmentation on our method. As shown in Fig. 7, without DNN augmentation (AF=0), even 15 negative models can not achieve high results. As the augmentation factor increase, the query accuracy difference (the difference between the query accuracy of positive models and negative models) increases greatly. With only 6 negative models and augmentation factor 5, our approach is able to achieve 93% query accuracy difference.

6 Conclusion

In this paper, We propose a robust DNN fingerprint method MetaFinger that fingerprints the inner decision area of the source model by meta-training. MetaFinger trains some models from scratch and augments models as meta-data. Then we optimize some images with meta-training guided by triple loss. Finally, we screen out the images that are only recognized by models derived from the source model as the query set. Compared with existing DNN watermark methods and DNN fingerprint methods, MetaFinger has the following advantages: 1) fidelity, it does not hurt the performance of source model; 2) robustness, it is able to identify all stolen models with 99.34% and 97.69% average query accuracy against six input transformations and three types of model modifications, which exceeds existing methods over 30%, 25% on CIFAR-10 and Tiny-ImageNet datasets. Therefore, our approach provides a new way to protect the IP of the owner's model.

Acknowledgments

This research was supported in part by the National Key Research and Development Program of China under No.2020YFB1805400, the National Natural Science Foundation of China (NSFC) under No. 61876134, the Fellowship of China National Postdoctoral Program for Innovative Talents under No.BX2021229, the Fundamental Research Funds for the Central Universities under No.2042021kf1030, the Natural Science Foundation of Hubei Province under No. 2021CFB089, the Open Foundation of Henan Key Laboratory of Cyberspace Situation Awareness under No. HNTS2022004.

References

[Adi *et al.*, 2018] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *USENIX*, pages 1615–1631, 2018.

[Ateniese *et al.*, 2015] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10(3):137–150, 2015.

[Cao *et al.*, 2021] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In *AsiaCCS*, pages 14–25, 2021.

[Chen *et al.*, 2019] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Blackmarks: Blackbox multibit watermarking for deep neural networks. *arXiv preprint arXiv:1904.00344*, 2019.

[Darvish Rouhani *et al.*, 2019] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *ASPLOS*, pages 485–497, 2019.

[Dziugaite *et al.*, 2016] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M Roy. A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*, 2016.

[Han *et al.*, 2015] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[Jia *et al.*, 2021] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *USENIX*, 2021.

[Le Merrer *et al.*, 2020] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32(13):9233–9244, 2020.

[Li *et al.*, 2019] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of dnn. In *ACSAC*, pages 126–137, 2019.

[Lukas *et al.*, 2019] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. *arXiv preprint arXiv:1912.00888*, 2019.

[Lukas *et al.*, 2021] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking?(extended version). *arXiv preprint arXiv:2108.04974*, 2021.

[Nguyen *et al.*, 2015] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436, 2015.

[Oh *et al.*, 2019] Seong Joon Oh, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer, 2019.

[Szegedy *et al.*, 2013] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[Szyller *et al.*, 2021] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. In *ACM MM*, pages 4417–4425, 2021.

[Uchida *et al.*, 2017] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *ICMR*, pages 269–277, 2017.

[Wang and Kerschbaum, 2021] Tianhao Wang and Florian Kerschbaum. Riga: Covert and robust white-box watermarking of deep neural networks. In *WWW*, pages 993–1004, 2021.

[Xie *et al.*, 2017] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[Xu *et al.*, 2017] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[Xu *et al.*, 2021] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. Detecting ai trojans using meta neural analysis. In *SP*, pages 103–120. IEEE, 2021.

[Zhang *et al.*, 2018] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *AsiaCCS*, pages 159–172, 2018.

[Zhang *et al.*, 2019] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *ICML*, pages 7472–7482. PMLR, 2019.