Abstract—Training highly performant deep neural networks (DNNs) typically requires the collection of a massive dataset and the use of powerful computing resources. Therefore, unauthorized redistribution of private pre-trained DNNs may cause severe economic loss for model owners. For protecting the ownership of DNN models, DNN watermarking schemes have been proposed by embedding secret information in a DNN model and verifying its presence for model ownership. However, existing DNN watermarking schemes compromise the model utility and are vulnerable to watermark removal attacks because a model is modified with a watermark. Alternatively, a new approach dubbed \textsc{DeepJ}udge was introduced to measure the similarity between a suspect model and a victim model without modifying the victim model. However, \textsc{DeepJ}udge would only be designed to detect the case where a suspect model’s architecture is the same as a victim model’s. In this work, we propose a novel DNN fingerprinting technique dubbed \textsc{DeepTaster} to prevent a new attack scenario in which a victim’s data is stolen to build a suspect model. \textsc{DeepTaster} can effectively detect such data theft attacks even when a suspect model’s architecture differs from a victim model’s. To achieve this goal, \textsc{DeepTaster} generates a few adversarial images with perturbations, transforms them into the Fourier frequency domain, and uses the transformed images to identify the dataset used in a suspect model. The intuition is that these adversarial images can be used to capture the characteristics of DNNs built on a specific dataset. To show the effectiveness of \textsc{DeepTaster}, we evaluated the detection accuracy of \textsc{DeepTaster} on three datasets (CIFAR10, MNIST, and Tiny-ImageNet) with three model architectures (VGG16, ResNet18, and DenseNet161) under various attack scenarios, including transfer learning, pruning, fine-tuning, and data augmentation. Overall, \textsc{DeepTaster} achieves a balanced accuracy of 94.95%, 94.95%, and 93.60% for CIFAR10, MNIST, and Tiny-ImageNet datasets, respectively, which are significantly better than 61.11% achieved by \textsc{DeepJ}udge in the same settings.

1. Introduction

Deep neural networks (DNNs) have recently gained much attention from academia and industry because they have proved useful in numerous applications, including image recognition [36], autonomous driving [22], and medical image classification [43]. One of the reasons for their success and widespread utilization in various domains is that IT giants such as Google, IBM, Microsoft, and OpenAI have released their pre-trained DNN models to the scientific community to promote further research and scientific advancement. In many cases, pre-trained models have been built on huge datasets collected, processed, organized, and labeled by the organisation.

Organisations that wish to commercialise the use of their proprietary DNN model can now do so via a cloud provider that offers Machine Learning as a Service (MLaaS). However, DNN models or datasets can potentially be stolen when they are used for Machine Learning as a Service (MLaaS) [29]. In particular, the dataset for MLaaS could be accessed and misused by a malicious insider. For example, a recent data breach incident on “Capital One” showed that an unauthorized insider attacker could access users’ data on the cloud server [24], demonstrating the possibility of dataset misuse from MLaaS providers – a malicious MLaaS provider can steal a proprietary dataset and use the dataset for her own DNN models without the dataset owner’s permission. Another possibility is the theft of a DNN model by external attackers by querying the model via MLaaS APIs. Recent studies (e.g., [25], [33], [41]) have shown that DNN model stealing attacks can effectively be launched even in real-world services. Therefore, it would be necessary for the DNN model owners to protect the intellectual property (IP) of their own models from stealing attacks.

Existing DNN IP protection mechanisms are categorized into DNN watermarking and DNN fingerprinting. DNN watermarking embeds the information of the model owner (i.e., watermark) into a proprietary model [1], [2], [6], [7], [9], [15], [30], [34], [42]. The model ownership can be verified by retrieving the identical or a similar watermark from a suspect model. There have been many proposals for developing effective DNN watermarking.
schemes. However, DNN watermarking has two limitations: (a) DNN watermarking is inherently invasive by design because this approach requires modifying the original DNN model to embed a watermark, which may change the DNN model's behavior [35], [42]. (b) DNN watermarking is not sufficiently resilient against adversarial attacks [39], [40]. Aiken et al. [3] showed that attackers could effectively manipulate neurons or channels in DNN layers that contribute to the embedded watermark for most state-of-the-art DNN watermarking schemes. Lukas et al. [20] recently demonstrated that transfer learning could remove nearly all of the tested 11 watermarking schemes.

Unlike DNN watermarking, DNN fingerprinting is non-invasive by design because this approach uses the unique characteristics (i.e., fingerprinting features) of each DNN model without modifying the model itself. A verifier can identify a model by examining its fingerprinting features [5], [21]. Generally, a single fingerprinting feature is insufficient to identify a model built through model stealing and adaptive attacks [8]. Chen et al. [8] recently introduced the state-of-the-art fingerprinting scheme dubbed DEEPJUDGE, which relies on multiple fingerprinting features to protect the copyright of a model. However, DEEPJUDGE uses fingerprinting features associated with the model's parameters. This indicates that DEEPJUDGE would not be effective in identifying the unauthorised use of the protected model's training dataset when a suspect DNN model is composed of different parameters or a different model architecture is used for the suspect DNN model. Additionally, in this paper, we found that DEEPJUDGE is not sufficiently effective in detecting models constructed through transfer learning [32], which is a method of reusing a pre-trained model for another task [40]. Our experimental results show that DEEPJUDGE's detection accuracy is significantly degraded for models built through transfer learning.

The state-of-the-art DNN fingerprinting scheme, DEEPJUDGE [8], is designed to detect the unauthorized use of a victim's DNN model where a suspect model's architecture is the same as a victim model's. Therefore, DEEPJUDGE would fail to detect the case where a victim's data is illegally used to build a suspect model whose architecture is different from the victim's original model architecture. To cover such an attack scenario (see Figure 1), we present a novel DNN fingerprinting scheme dubbed DEEPTASTER.

![Figure 1. New attack scenario in which a victim's dataset is stolen to build a suspect model.](image)

In this paper, we show that the characteristics of a specific dataset used to build a DNN model can be uniquely determined with the spectra of the gradient-based adversarial examples in terms of the decision boundaries of a target model. Interestingly, adversarial examples generated for different DNN models, which were all trained on the same dataset, show statistically similar patterns in the Discrete Fourier Transform (DFT) domain, even when their model architectures are different. Motivated by these findings, we propose DEEPTASTER as a scheme to detect data theft attacks (see Figure 1). DEEPTASTER generates a few adversarial images with perturbations, transforms them into the DFT domain, and uses their statistical properties as the features of a meta-classifier to identify the dataset used in a suspect model. According to our experimental results, DEEPTASTER can highly accurately identify a dataset used to build a suspect model even when the suspect model's architecture differs from a victim model's. To the best of our knowledge, DEEPTASTER is the first attempt to detect this new type of model stealing attack.

We summarize our key contributions as follows:

- We propose a novel DNN fingerprinting scheme, DEEPTASTER, particularly to detect data theft attacks. DEEPTASTER uses a meta-classifier to determine whether a suspect model is built on a proprietary dataset within a small number of queries (see Section 4).
- We introduce six new attack scenarios, including multi-architectures, data augmentation, retraining, transfer learning, fine-tuning, and pruning in which a victim’s data is stolen to build a suspect model – a malicious cloud service provider or insider attacker can steal user data and use them to build her own model (see Section 3). Our experimental results demonstrate that the state-of-the-art DNN fingerprinting scheme, DEEPJUDGE, would be ineffective in preventing these attacks, especially when a suspect model architecture differs from a victim’s original model architecture. We discuss the root cause of DEEPJUDGE's limitation in detecting data theft attacks (see Section 6).
- We comprehensively evaluate the effectiveness of DEEPTASTER under the six attack scenarios with three datasets (CIFAR10, MNIST, and Tiny-ImageNet) and three model architectures (VGG16, ResNet18, and DenseNet161). Overall, DEEPTASTER achieves a balanced accuracy of 94.95%, 94.95%, and 93.60% for CIFAR10, MNIST, and Tiny-ImageNet datasets, respectively, which outperformed DEEPJUDGE in the same settings (see Section 5).

2. Background

This section provides the background of deep neural networks, adversarial perturbations, and Discrete Fourier Transform (DFT).

2.1. Deep Neural Networks (DNNs)

A DNN classifier is a function \( f : X \rightarrow Y \) that maps the input \( x \in X \) to the probability \( y \in Y \) that the input belongs to each class [38]. DNN classifier consists of \( L \) layers \( \{l_0, l_1, ..., l_L\} \), each of which is a set of neurons \( \{n_{L,0}, n_{L,1}, ..., n_{L,N_L}\} \).

Here, the first layer \( l_0 \) is called the input layer, the last layer \( l_L \) is called the output layer,
and the rest $l_2, \ldots, l_{L-1}$ are called the hidden layers. The parameters within hidden layers are called weights and biases. The neurons that compose each layer calculate the output by applying a linear function followed by a non-linear function called the activation function to the input sequentially. We then apply a softmax activation function $\sigma(\cdot)$ to output layer $f_L(\cdot)$ to convert likelihoods into probabilities for each predicted class. Training the above DNN classifier requires a loss function that can be optimised by gradient descent on all trainable weights and biases. An example of loss function is cross-entropy.

### 2.2. Adversarial Perturbation and Attack

In the computer vision domain, an adversarial perturbation is a maliciously crafted perturbation of the input sample (image) that can lead to misclassification [11], [23] by the model. One known perturbation-generating mechanism is gradient-based adversarial attacks, such as the fast gradient sign method (FGSM) [11]. FGSM generates a minimal random modification to the input image in the direction that affects the target classifier prediction. A “small modification” (perturbation), for instance, changing a single pixel’s color, may be enough to fool the model decision boundaries. We observe that adversarial algorithms craft the perturbation in correlation with the DNN dataset ownership IP within hidden layers and thus likely carry sufficient information of the learned knowledge to be used as an IP protection mechanism. We use foolbox [26], a standard library that implements various adversarial attacks. Finally, we select FGSM as the best option.

FGSM is a gradient-based adversarial algorithm proposed by Goodfellow [11]. Assuming the original image is $x$, $\nabla$ is the slight permutation applied to $x$ that produces the adversarial sample $\bar{x}$. The training process starts with the goal of maximizing the loss function $J(x,y)$ to obtain the adversarial sample $\bar{x}$. Maximizing $J$ means the noise-added samples no longer belong to class $y$, thus accomplishing the goal. In the entire optimisation process, the $L_\infty$ constraint $\|\bar{x} - x\|_\infty \leq \epsilon$ must be satisfied. In summary, the FGSM adversarial examples can be obtained by the following equation:

$$\bar{x} = x + \epsilon \cdot \text{sgn} (\nabla_y J(f(x), y))$$ (1)

### 2.3. Discrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) transforms a sequence of numbers $\{x_0, x_1, \ldots, x_N\}$ in the time domain into another sequence of numbers $\{y_0, y_1, \ldots, y_N\}$ in the frequency domain using the equation $y_k = \sum_{n=0}^{N} x_n \cdot e^{- \frac{2\pi}{N} kn}$. Applying DFT to an image allows the spectrum, which is the intensity of each frequency component, to be represented like an observing. Observing the spectrum of an image allows us to gather more concentrated noise information that reflects the DNN dataset ownership IP. The intuition is that we aim to leverage those DFTs to track the dataset ownership IP across architectures.

This image processing technique is already widely known, and various methods, such as the Fast Fourier Transform (FFT), have been proposed to quickly generate the Fourier transform of the image.

### 3. Threat Model

For the evaluation of DEEPTASTER, we assume different levels of adversarial settings to execute a DNN IP stealing attack. In all scenarios, the adversary aims to steal the dataset ownership IP either from the dataset itself or from the DNN model trained on it.

**Overview.** We consider the leakage of the dataset or and the DNN model. From Dataset-perspective, it has been shown that the MLaaS ecosystem enables dataset access and misuse by malicious insiders, as shown recently in “Capital One” data breach incident. With the aim to avoid IP violation detection, an adversary may (a) use the leaked dataset and use it to train on a different DNN architecture, or (b) augment the leaked dataset with more samples before training. We are not aware of any existing work that address these dataset intelligence IP violations. On the other hand, from the DNN model perspective, an adversary can steal models from the victim’s private cloud or execute a model extraction attack using the MLaaS API of the victim model. In the former case, the adversary can fine-tune, prune, and transfer learn the stolen model to increase performance and to hide the fact that they were stolen. In addition, the adversary might commercially use DNN models released for education, or leak models they have stolen from the private cloud. We designed and tested the following threat models.

**Assumptions.** We consider the following assumptions. (a) Capacity: the adversary could steal the dataset and/or the model. (b) Goal: the adversary aims to steal intelligence of the dataset and fool the copyright verification. (c) Assumption: the surrogate model developed by the adversary is well-trained, with sufficient accuracy that the adversary stands to profit. (d) Model: the adversary could steal the DNN model. From Dataset-perspective, it has been shown that the MLaaS ecosystem enables dataset access and misuse by malicious insiders, as shown recently in “Capital One” data breach incident. With the aim to avoid IP violation detection, an adversary may (a) use the leaked dataset and use it to train on a different DNN architecture, or (b) augment the leaked dataset with more samples before training. We are not aware of any existing work that address these dataset intelligence IP violations. On the other hand, from the DNN model perspective, an adversary can steal models from the victim’s private cloud or execute a model extraction attack using the MLaaS API of the victim model. In the former case, the adversary can fine-tune, prune, and transfer learn the stolen model to increase performance and to hide the fact that they were stolen. In addition, the adversary might commercially use DNN models released for education, or leak models they have stolen from the private cloud. We designed and tested the following threat models.

1. **Multi-Architecture Attack (MAA).** The adversary steals the victim’s dataset and uses it to train a model with architecture that’s different to the original victim model. None of the existing IP protection using fingerprinting or watermark schemes have considered this attack.

2. **Data Augmentation Attack (DAA).** The attacker in this case steals the victim’s dataset. Then they create a new dataset by combining the stolen data with data from the same domain, with the aim to either hide the stolen data, or to achieve a better model learning rate. The attacker trains a different DNN model based on the combined dataset or transfer learning from a stolen pretrained model on the victim dataset into the combined dataset and uses it commercially. For each case, the attackers’ model has some dataset intelligence obtained from the stolen dataset. Here, our goal is to show that DEEPTASTER can detect that dataset intelligence obtained from the stolen dataset.
(3) Model Retraining Attack (MRA). The adversary has part of the victim’s dataset. They also know the structure of the victim’s model. The adversary uses the dataset they have to retrain a model of the same structure as the victim’s model in order to avoid IP detection and then use the retrained model commercially.

(4) Transfer Learning Attack (TLA). The adversary steals the victim’s model. Then the adversary uses transfer learning to fine-tune the model on another dataset that the adversary has, in order to use the stolen model in another field. This neutralizes various attempts to detect the model is stolen, and allows the model to work in the desired domain.

(5) Model Fine-tuning Attack (MFA). The adversary knows the structure and parameters of the victim’s model. They also have a portion of the dataset used by the victim for model training. To conceal the fact that the model was stolen, the adversary fine-tunes the model on the portion of the dataset that they possess, and then use it commercially.

(6) Model Pruning Attack (MPA). The adversary has the victim’s model. However, the adversary does not have any information on the dataset used for training. The adversary aims to prune and redistribute the stolen model.

4. DEEPASTER System Design

In this section, we present DEEPASTER, a dataset IP tracking tool that verifies whether an attacker’s model has stolen knowledge from a victim’s dataset or model. We first discuss the design requirements before presenting the system design overview of DEEPASTER. We then deep dive into the major three components of the system: adversarial perturbation generation and transformation, meta-classifier generation, and verification.

Design Requirements. Protecting dataset IP presents several challenges compared to protecting model-dependent IP which may be solved by existing watermark and fingerprinting schemes. To solve its unique challenges, we identify the following criteria for a reliable copyright protection and verification method for protecting the dataset IP.

1) Robustness. The protection should capture the dataset ownership IP and be resilient to model architecture change. To the best of our knowledge, this is the first work that tackle this design challenge. The protection should also be generalisable to ensure robustness even when applied to protect various datasets.

2) Fidelity. The ownership protection and verification process should not impact the normal model utility.

3) Efficacy. The verification should have high accuracy and recall in detecting stolen dataset intelligence, even across multiple model architectures.

4) Efficiency. The verification process should be efficient and lightweight, e.g., taking only a few samples to verify.

4.1. DEEPASTER Overview

As depicted in Figure 2, DEEPASTER consists of the following 3-step process: (a) the generation of adversarial perturbation samples and their translation to the Fourier frequency domain using Discrete Fourier Transform (DFT), (b) the creation of a meta-classifier that is trained on the spectra (i.e., DFT samples) in order to distinguish the dataset intelligence, and (c) verification of the suspect model by generating adversarial perturbation samples from it and then testing them using the meta-classifier.

Details of each step are described in the following subsections.

Algorithm 1 Adversarial Perturbation Generation and Transformation.

```
Input: Sample image I and target model M
Output: Adversarial DFT image Adv
1: procedure GenerateAdv(D,I)
2: \text{Adv}_\text{raw} \leftarrow \text{FGSM}(M,I)
3: \text{Adv}_\text{per} \leftarrow \text{Adv}_\text{raw} - I
4: \text{Adv}_\text{Fourier} \leftarrow \text{FourierTransform}(\text{Adv}_\text{per})
5: \text{Adv} \leftarrow \text{ShiftLog}(\text{Adv}_\text{Fourier})
6: return Adv
7: end procedure
```
4.1.2. Meta-Classifier Generation. To ensure robust dataset intelligence characteristics are captured, we develop a one-class meta-classifier that is trained on adversarial DFT images generated from multiple model architectures, each of which is trained on the victim dataset. The intuition here is to build a resilient detector that can efficiently recognize the stolen dataset intelligence, even when the adversary changes the model architecture or transfers the intelligence to other models as an adaptive attack strategy. We choose a Deep Support Vector Data Description (DeepSVDD) [27] model as a meta-classifier from the various types of one-class classification models. SVDD [31] tries to extract the common characteristics of data variation to conduct the classification. In particular, DeepSVDD trains a neural network to minimize the volume of a hypersphere that encloses the network representations of the data. Therefore, we utilize this mechanism

**Algorithm 2 Meta-classifier Generation.**

**Input:** The subset of victim dataset $D'$, victim models $M_1, ..., M_n$ trained on victim dataset $D$.

**Output:** Meta-classifier $Model_{meta}$, Threshold $\tau$

1. Split $D'$ into $D_{train}', D_{val}', D_{test}$
2. $Adv_{train} \leftarrow \bigcup_{k=0}^{n-1} GenerateAdv(M_k, D_{train}')$
3. $Adv_{val} \leftarrow \bigcup_{k=0}^{n-1} GenerateAdv(M_k, D_{val}')$
4. Train $Model_{meta}$ on $Adv_{train}$
5. $output \leftarrow Model_{meta}(Adv_{val})$
6. Sort $output$
7. $\tau \leftarrow output[0.04 + \text{length}(output)]$
8. return $Model_{meta}$ and $\tau$
of DeepSVDD to extract the common fingerprint across the different models trained on the same dataset.

Algorithm 2 describes the procedure to generate the DeepTaster meta-classifier. The input to our algorithm is a subset, \( D' \), of the victim dataset \( D \) (\(|D'| \ll |D|\)), as well as the \( n \) different victim models \( M_1, ..., M_n \) trained on the victim dataset \( D \). The output is the meta-classifier \( Model_{meta} \) and the corresponding decision threshold \( \tau \). First we split the sub-dataset \( D' \) into a training dataset \( D'_{train} \), validation dataset \( D'_{val} \), and test dataset \( D'_{test} \). The training dataset is used to train the meta-classifier, the validation dataset \( D'_{val} \) is used to calculate the threshold \( \tau \), and \( D'_{test} \) is used to evaluate the suspect model. The training dataset \( D'_{train} \) and validation dataset \( D'_{val} \) are used to generate the adversarial samples \( Adv_{train} \) and \( Adv_{val} \) respectively from the victim models and the sub-dataset \( D' \) using \( GenerateAdv() \) as in lines 1-3. The second step is to train the one-class classifier on the training adversarial samples \( Adv_{train} \), as in line 4. The third step is to define a threshold value using the output of the meta-classifier on the validation adversarial samples \( Adv_{val} \), as in lines 5-7. The classification decision threshold is selected so as to balance the true positive and true negative rate. Specifically, the threshold is chosen that 96% of the validation set’s samples lie below the meta-classifier’s threshold and therefore the misclassified validation samples account for at most 4%. The selected threshold is used for classifying the suspect model based on the measurement of adversarial DFT samples generated from it via the meta-classifier. The details of the model verification process are given in Section 4.1.3. Note that the threshold is meta-classifier dependent, instead of suspect model dependent. The more victim models with variant architectures are used to train the meta-classifier, the more knowledge it acquires. As a result, the threshold value might be slightly different, even though the victim dataset is the same. Furthermore, the threshold could also be adaptively adjusted for different preferences.

Algorithm 3 Validation using DeepTaster.

\[
\text{Input:} \text{Meta-classifier } Model_{meta}, \text{the threshold } \tau, \text{the test dataset } D'_{test}, \text{and the suspect model } S. \\
\text{Output:} \text{Verification results}
\]

1. \( Adv_{test} \leftarrow GenerateAdv(S, D'_{test}) \)
2. \( X \leftarrow 0 \)
3. \( k \leftarrow \text{len}(Adv_{test}) \)
4. \( \text{while } k \neq 0 \text{ do} \)
5. \( X \leftarrow X + Model_{meta}(Adv_{test}[k]) \leq \tau \)
6. \( k \leftarrow k - 1 \)
7. \( \text{end while} \)
8. \( \text{if } X > \text{len}(Adv_{test}) \times \frac{1}{2} \text{ then} \)
9. \( S \) is stolen model
10. \( \text{else} \)
11. \( S \) is benign model
12. \( \text{end if} \)

4.1.3. Verification. Algorithm 3 describes the verification procedure using DeepTaster. The input to our algorithm is the meta-classifier \( Model_{meta} \), the threshold value \( \tau \), the test dataset \( D'_{test} \), and the suspect model \( S \). The datasets \( D'_{train} \) and \( D'_{val} \), which are used in Algorithm 2, and \( D'_{test} \) are subsets of \( D' \) with no intersection, so that there is no bias in the validation and testing steps. The output is the verification result, indicating whether the suspect model is stolen or not — i.e., contains stolen dataset intelligence from a victim dataset. To test the suspect model, we generate the test adversarial DFT samples using the steps in Section 4.1.1 and feed the output to the meta-classifier one-by-one, as in lines 1-7. If more than half of the samples fall below the classifier’s threshold, it means more than half of samples are discerned as stolen, and the suspect model is decided to be stolen, as in lines 8-12. Figure 5 shows the CIFAR10 meta-classifier’s threshold value and the results of the CIFAR10 validation and test sets versus the ImageNet test set. Given the protected dataset CIFAR10, Figure 5 demonstrates that our meta-classifier is capable of distinguishing suspect models (trained on the validation set of CIFAR10, carrying the intelligence of CIFAR10) from benign models (trained on the Imagenet), with high accuracy and across a variety of model architectures (DenseNet, ResNet, and VGG).

5. Experiments

We implemented DeepTaster as a self-contained toolkit in Python. In this section, we evaluate the performance of DeepTaster against an extensive list of six different attacks mentioned in Section 3. Some of these attacks, such as fine-tuning and pruning, are well studied in watermarking. We also examine DeepTaster against more challenging adaptive attack scenarios such as transfer learning, retraining, and the most challenging - multi-architecture - which has never been considered before in the literature. To ensure the generalizability of DeepTaster, we generate three meta-classifiers which track CIFAR10, MNIST, and Tiny-ImageNet respectively. We also compare our results to the best state of the art fingerprinting technique named DeepJudge [8].

5.1. Experimental Setup

Datasets and Victim Models. We use four datasets including CIFAR10 [16], MNIST [19], Tiny-ImageNet [17], and ImageNet [10]. The first three datasets are used as victim datasets — where they are used to train a meta-classifier to be able to track each dataset respectively. The
ImageNet dataset is used to check the True Negative Rate (TNR). All datasets are image classification datasets with a varying number of classes, ranging from 10 classes in CIFAR10 and MNIST to up to 1000 in ImageNet, as described in Table 2. We point out that we use only half of the Tiny-ImageNet dataset (i.e., 100 classes) for running the experiments in order to reduce the experimental compute time.

### Table 2. Experiment dataset.

| Dataset          | # Classes | Usage           |
|------------------|-----------|-----------------|
| CIFAR10          | 10        | Victim / Suspect|
| MNIST            | 10        | Victim / Suspect|
| Tiny-ImageNet    | 100*      | Victim / Suspect|
| ImageNet         | 1000      | Suspect         |

We use three commonly used DNN architectures to train the victim models on each of the victim datasets, including the VGG16 [28], ResNet18 [13], and DenseNet161 [14]. The details of each model are described in Table 3. We note that Tiny-ImgNet based models accuracy is fairly low. However, we use it as a generalisability use case to investigate that could we still track the propriety of proportion of large datasets like ImageNet use in deep neural networks.

### Table 3. Datasets, Models, parameters we used and their baseline accuracy.

| Dataset       | Architecture | # Params | Accuracy   |
|---------------|--------------|----------|------------|
| CIFAR10       | VGG16        | 134301514| 81.67%     |
|               | ResNet18     | 11181642 | 72.98%     |
|               | DenseNet161  | 26494090 | 76.80%     |
| MNIST         | VGG16        | 134301514| 99.28%     |
|               | ResNet18     | 11181642 | 99.17%     |
|               | DenseNet161  | 26494090 | 99.03%     |
| Tiny-ImageNet | ResNet18     | 11181642 | 40.09%     |
|               | DenseNet161  | 26494090 | 51.12%     |

### 5.2. Meta-Classifier Evaluation Settings

#### Training Configuration. We create a meta-classifier that tracks the knowledge of a victim dataset using the method presented in Section 4.1.2. We generate 2176 adversarial DFT images for each victim model then divide that into train/val/test datasets as follows — 1600 images as the training set for the meta-classifier, 288 images as the validation set to obtain the classifier thresholds, and the remaining 288 images to conduct evaluation of verification. In this case, the threshold has been set so that 96 percent of validation samples fall below the threshold. The meta-classifiers balanced accuracy, i.e., (true positive + true negative)/2, is 94.00%, 95.40%, and 94.47% for CIFAR10, MNIST, and Tiny-ImageNet respectively. This indicates the reliability of using the meta-classifier to detect the existence of dataset intelligence within a suspect model.

#### Metrics. We calculate three metrics: True Positive Rate (TPR), True Negative Rate (TNR), Balanced Accuracy (BA), and compute the Area Under the Receiver Operating Characteristic curve (ROC AUC) score. TPR means the ratio of right answer (i.e., detecting stolen model as “Stolen”), when we test 288 adversarial samples of a stolen model with Meta-classifier. TNR means the ratio of right answer (i.e., labelling benign model as “Benign”), when test 288 adversarial samples of a benign model. Balanced Accuracy (BA) is calculated as the average of the TPR and TNR. Using both adversarial samples of a stolen and benign model, the ROC AUC is calculated.

### 5.3. Defending Against Various Data IP Attacks

In the following, we focus on the feasibility of our DEEP TASTER against the six attack scenarios presented in the threat model in Section 3. In Section 5.3.1, we check the performance of three meta-classifiers against Multi-Architecteure Attack (MAA). In other five attacks, without loss of generality, the meta-classifier in this section is built to protect the dataset intelligence of CIFAR10 across those six attacks. We consider the generalisability of DEEPTASTER in protecting two other datasets in Section 5.5.

#### 5.3.1. Multi-Architecture Attack (MAA).

**Attack Strategies.** We evaluate DEEP TASTER against MAA to investigate if DEEP TASTER can detect whether the suspect model contains dataset intelligence from our stolen dataset. Here the attacker trains the stolen dataset on multiple different model architectures to subvert IP detection. We select three victim datasets as CIFAR10, MNIST, and Tiny-ImageNet and train three different architectures (VGG16, ResNet18, and DenseNet161) for each dataset respectively. These models trained on ImageNet [10] are used as the benign case for MNIST and CIFAR10. For each case, we target one dataset as victim across the three models and the other two datasets as benign across the same 3 models. Table 3 shows the accuracy of those models. To calculate the TPR and TNR of DEEP TASTER, we generate 288 adversarial samples with 12 models trained on four different datasets and three different architectures. Then, we use Algorithm 3 to test the adversarial DFT samples against the meta-classifier we built to detect stolen intelligence from the victim dataset.

**Efficacy.** As shown in Table 4, our DEEP TASTER exhibits high efficacy against MAA in both stolen and benign scenarios regardless of the victim datasets. Our DEEP TASTER can distinguish all models with at least 64% accuracy. The BA of the meta-classifier of each of CIFAR10, MNIST, and Tiny-ImageNet shows high performance at 94.95%, 94.95%, and 93.60%.

**Remark 1:** DEEP TASTER is effective and efficient in identifying cross-architecture dataset intelligence copies.

#### 5.3.2. Data Augmentation Attack (DAA).

**Attack Strategies.** Here the target stolen dataset is CIFAR10, and we assume that the attacker creates a CIFAR15 dataset by adding extra five classes of images from CIFAR100 dataset to claim a different dataset from CIFAR10. Such a strategy aims to obtain better model utility while bypassing IP verification for stealing the intelligence of CIFAR10 dataset. We select the following 5 random classes from CIFAR100 that are not within
CIFAR10: ‘apples’, ‘bicycle’, ‘can’, ‘roses’, and ‘clock’. We consider two attack cases. (a) The attacker uses a stolen pre-trained ResNet model that was trained on the target dataset CIFAR10, and then further fine-tunes that model on the CIFAR15 dataset they have created. (b) The attacker trains a model such as ResNet from scratch on the CIFAR15 dataset. In both cases, we investigate how DeepTaster performs against those two attacks at various epochs (20, 60, and 100). We also use MNIST dataset trained on the same model as the benign case. The mean accuracy value of these attack models is about 72.48% (see Table 5 for complete accuracy results).

Efficacy. As presented in Table 5, DeepTaster is capable of detecting that the suspect models contain stolen knowledge from the victim dataset CIFAR10. For the first scenario where the attacker transfers learn from a pretrained stolen model CIFAR10, the average and SD of the TPR is 71.53% and 7.11. For the second scenario where the attacker trains a model from scratch, the average and SD of the TPR is 68.17% and 7.39. While DeepTaster has accurately detected all cases as stolen with 69.85% mean accuracy, training from scratch as attack strategy seems more challenging and lower the detection rate especially when using high number of epochs. However, it also means the attacker would compromise the utility aspect by lowering the model accuracy as well.

### 5.3.4. Transfer Learning Attack (TLA)

**Remark 2:** Data Augmentation Attacks are more challenging in general especially when the attacker trains a model from scratch; however, DeepTaster still correctly identifies that the new model contains stolen dataset intelligence.

### 5.3.3. Model Retraining Attack (MRA)

**Attack Strategies.** In MRA, an attacker trains the ResNet18 model on 10%, 30%, 50%, 70%, 90%, and 100% of the CIFAR10 dataset. We split the dataset uniformly — including an equal number of samples from every class. The attacker aims to steal the dataset to build the attacker’s model while evading the data theft attack detection. We evaluate the TPR and trained model accuracy every 50 epochs up to 200. Since MRA experiment results vary depending on the random seed initialization value, we repeat these experiments three times and report the average results. We also evaluate the TNR with the ResNet18 model on 10%, 30%, 50%, 70%, 90%, and 100% of the MNIST dataset with every 5 epochs up to 20.

**Efficacy.** As shown in Table 6, the results demonstrate that when the portion of the stolen dataset that is used for training is ≥70%, DeepTaster is capable of detecting that the new trained suspect model contains a stolen dataset intelligence. Despite fluctuations in detection accuracy as the number of epochs varies, it is clear that the lowest TPR is 55% and reaches 81.48% when 100% of the dataset used in the training. On the other hand, when the attacker steals <70% of the dataset, we might not be able to detect the suspect model as violating the copyright. This might also means the attacker might not be interested in the dataset intelligence as whole, but only target to steal certain portion of the samples. Likewise, our DeepTaster can detect benign models trained on ≥70% of MNIST dataset as benign with TNR of 64.24%.

### 5.3.4. Transfer Learning Attack (TLA)

**Remark 3:** Retraining attacks have never been considered before due to their major manipulation of the model parameters, which presents challenges in detecting copyright infringements. DeepTaster shows a decent capability in detecting retraining attacks when ≥70% of the stolen dataset is used in the training.

---

**TABLE 4. MAA results for CIFAR10, MNIST, Tiny-ImageNet meta-classifiers. The copy field values below indicate the classification results (Yes indicates ‘Stolen’) and (No indicates ‘Benign’). The [GREEN] indicates correct classification.**

| Victim | Ground Truth | Suspect | ResNet | VGG | DenseNet | Copy? | Balanced Accuracy (%) |
|--------|--------------|---------|--------|-----|---------|-------|---------------------|
| CIFAR10 | Stolen | CIFAR10 | 95.14 | 90.97 | 96.53 | Yes | 94.95 |
| | | MNIST | 89.58 | 100 | 81.6 | No | |
| | | Tiny-ImageNet | 87.85 | 85.76 | 97.57 | No | |
| | | ImageNet | 99.31 | 100 | 98.61 | No | |
| | Benign | | | | | | |
| | | Tiny-ImageNet | 76.39 | 98.96 | 99.65 | No | |
| | | ImageNet | 100 | 64.93 | 100 | No | |
| | MNIST | Stolen | 93.75 | 97.57 | 98.61 | Yes | 94.95 |
| | | Tiny-ImageNet | 100 | 100 | 89.58 | No | |
| | | ImageNet | 100 | 64.93 | 100 | No | |
| Tiny-ImageNet | Stolen | Tiny-ImageNet | 94.44 | 96.53 | 99.31 | Yes | |
| | | MNIST | 100 | 73.26 | 83.33 | No | |

**TABLE 5. DAA results for CIFAR10 meta-classifier. The copy field values below indicate the classification results (Yes indicates ‘Stolen’) and (No indicates ‘Benign’). [GREEN] indicates correct classification.**

| ResNet Model | Epochs | Detection Acc. | Model Acc. | Copy? |
|--------------|--------|----------------|------------|-------|
| TPR% Positive Pretrained CIFAR10 | 20 | 63.19 | 72.9 | Yes (3/3) |
| 60 | 80.56 | 72.74 | |
| 100 | 70.83 | 71.93 | |
| TPR% Positive Scratch CIFAR10 | 60 | 63.24 | 72.87 | Yes (3/3) |
| 100 | 82.64 | 74.63 | |
| TNR% Negative Pretrained MNIST | 20 | 85.02 | 99.60 | No (3/3) |
| 60 | 79.66 | 99.69 | |
| 100 | 99.65 | 99.68 | |
| TNR% Negative Scratch MNIST | 20 | 66.87 | 99.54 | No (3/3) |
| 60 | 86.11 | 99.67 | |
| 100 | 100 | 99.67 | |
**Remark 5:** Our DEEPASTERT is robust against MFA regardless of the size of training dataset or the number of training epochs.

### 5.3.6. Model Pruning Attack (MPA).

**Attack Strategies.** In this scenario an attacker prunes 20%, 40%, and 60% of the victim’s ResNet18 model that is trained on the target dataset CIFAR10. Then the attacker fine-tunes the model for 5 epochs with a small learning rate of 0.00005. To evaluate the TNR, we also perform the same pruning and fine-tuning to the benign ResNet18 model trained on MNIST. We ensure that all MPA and benign ResNet models have decent accuracy.

**Efficacy.** As exhibited in Table 9, the TPR of the pruned model is 98.26%, 84.72% and 85.07% as the percentage of the parameters pruned increases from 20 to 60. As many parameters were pruned, the accuracy of the model decreased slightly, and thus the TPR decreased. Nevertheless, DEEPASTERT provides a high detection TPR at 93.85% (6.30) on average. On the other hand, the TNR is 99.31%, 99.31%, and 100%, which is uniformly high regardless of how many parameters are pruned. In the case of ROC AUC, the obtained results shows high performance at 0.9445.

**Remark 6:** While an increase in the percentage of neurons pruned may result in a slightly lower detection accuracy, nevertheless DEEPASTERT is still robust against MPA.

### 5.4. Comparison with Existing Fingerprinting Techniques

**Comparison Settings.** We conduct experimental comparisons with DEEPJUDGE [8], the leading fingerprinting technique. DEEPJUDGE generates four metrics for white-box evaluation and two metrics for black-box evaluation. It uses majority voting where 3 out of 4 metrics have to produce values < threshold to support the correct final judgement of being stolen. DEEPJUDGE has been designed to provide architecture dependent protection — namely, all model parameters need be the same including the number of classes, etc. On the other hand, our DEEPASTERT is designed to be architecture agnostic to enable the dataset intelligence to be tracked even when the model architecture is changed. Hence, DEEPJUDGE is not able to detect MAA as stolen models owing to its design limitation. Therefore, we compare DEEPJUDGE with...
TABLE 8. MFA results for CIFAR10 meta-classifier. The copy fields values below indicate the classification results (Yes indicates “Stolen”) and (No indicates “Benign”). GREEN indicates correct classification.

| ResNet18 Model | Epochs | Model Acc. | Detection Acc. | Copy? |
|----------------|--------|------------|----------------|-------|
| Positive CIFAR10 | TPR% | 89.24 | 96.38 | Yes (4/4) |
| 1              | 90.01 | 96.53 | 73.91         |
| 20             | 90.49 | 96.94 | 73.91         |
| 50             | 90.98 | 96.94 | 73.91         |
| 100            | 91.47 | 96.94 | 73.91         |
| 200            | 91.96 | 96.94 | 73.91         |

TABLE 9. MPA results for CIFAR10 meta-classifier with three positive models and three negative models. The copy field values below indicate the classification results (Yes indicates “Stolen”) and (No indicates “Benign”). GREEN indicates correct classification.

| ResNet18 Model | Prune % | Model Acc. | Detection Acc. | Copy? |
|----------------|---------|------------|----------------|-------|
| Positive CIFAR10 | TPR% | 70.63 | 96.28 | Yes (3/3) |
| 20             | 71.63 | 96.28 | 96.28         |
| 40             | 72.70 | 96.28 | 96.28         |
| 60             | 73.78 | 96.28 | 96.28         |
| 80             | 74.78 | 96.28 | 96.28         |
| Negative MNIST | TNR% | 99.44 | 92.71 | No (3/3) |
| 20             | 99.44 | 92.71 | 92.71         |
| 40             | 99.44 | 92.71 | 92.71         |
| 60             | 99.44 | 92.71 | 92.71         |
| 80             | 99.44 | 92.71 | 92.71         |

5.5. Generalisation Efficacy

To examine the generalisability of DeepTaster, we conduct evaluations using two other datasets: Tiny-ImageNet and MNIST. For Tiny-ImageNet, we experiment with the four most challenging attack scenarios: MAA, DAA, MRA, and MFA. Both MAA and DAA examine the robustness when the model architecture is changed or the number of output classes is altered. For MAA, we use DenseNet, ResNet, and VGG. For DAA, the attacker starts with a model that is pre-trained on the 100-class Tiny-ImageNet dataset. The attacker then creates a dataset of 110 classes by adding 10 more classes of Tiny-ImageNet that were not used in the initial victim model training, and fine-tunes the model on the new 110-class dataset. The other two attacks are the MRA and MFA to investigate the resilience of DeepTaster against retraining and fine-tuning. Figure 6 shows all the TPR and TNR results of Tiny-ImageNet. It is clear that our DeepTaster is robust against all attacks and is able to detect them with high TPR and TNR. The only exception is when MRA is executed using a low percentage (<50%) of the dataset, in which case our DeepTaster could not detect that as “Stolen.” (which is consistent with the Remark 3).

Similarly for the MNIST dataset, we conduct experiments to further examine the generalisability and resilience of our DeepTaster on a diverse range of datasets. For DAA, the MNIST has only 10 digits from zero to nine, so we take a pre-trained MNIST model and augment the MNIST samples within each class using another dataset from the same domain named EMNIST. The results of the 4 attacks are shown in Figure 7. It is obvious that despite changing the target dataset that needs to be tracked to MNIST, our DeepTaster is robust in general against all the attacks with high TPR and TNR. Our retraining attack results are consistent with the earlier findings that when a low percentage of the dataset is used, our DeepTaster could not identify the retrained model as “Stolen.” Still, our TNR is very high which indicates we have no challenge to identify the benign cases.

6. Discussion

Has DeepTaster met our target design requirements? The obtained results demonstrate that Deep-
TABLE 10. DATA THEFT ATTACK DETECTION RESULTS OF DEEPJUDGE, THE LEADING STATE OF THE ART FINGERPRINTING TECHNIQUE.
DEEPJUDGE USES MAJORITY VOTING, WHERE 3 OUT OF 4 METRICS HAVE TO PRODUCE VALUES < THRESHOLD TO SUPPORT THE RIGHT FINAL JUDGEMENT OF BEING STOLEN. GREEN INDICATES CORRECT CLASSIFICATION, AND RED INDICATES MISCLASSIFICATION.

| Victim          | Ground Truth | Suspect | Metric1 | Metric2 | Metric3 | Metric4 | Copy? | TPR / TNR (%) |
|-----------------|--------------|---------|---------|---------|---------|---------|-------|----------|
| **CIFAR10**     |              |         |         |         |         |         |       | 100 (TPR) |
| Stolen          |              | CIFAR10 | 0.0019  | 0.0111  | 0.2370  | 0.2828  | Yes   |           |
|                 |              | CIFAR10 DAA | 0.0183  | 0.1364  | 0.2558  | Yes    |       |           |
|                 |              | CIFAR10 MFA | 0.0085  | 0.0135  | 0.1185  | 0.2558  | Yes    |           |
|                 |              | CIFAR10 TLA | 0.1052  | 0.1844  | 0.3444  | Yes    |       |           |
|                 |              | CIFAR10 MFA | 0.0012  | 0.0074  | 1.4444  | 1.4444  | No    |           |
|                 |              | CIFAR10 MFA | 0.0012  | 0.0074  | 1.4444  | 1.4444  | No    |           |
|                 |              | MNIST MFA | 0.0018  | 0.0121  | 0.4991  | 0.5804  | Yes   |           |
|                 |              | Tiny-ImageNet | 0.0004  | 0.0059  | 1.2621  | 1.2777  | Yes   |           |
| **MNIST**       |              |         |         |         |         |         |       | 0 (TNR)  |
| Stolen          |              | MNIST   | 0.0004  | 0.0183  | 0.1364  | 0.2558  | Yes   |           |
|                 |              | MNIST DAA | 0.0286  | 0.1173  | 1.5526  | 1.8548  | Yes   |           |
|                 |              | MNIST MFA | 0.1874  | 0.0183  | 0.1364  | 0.2558  | Yes   |           |
|                 |              | MNIST MFA | 0.0018  | 0.0121  | 0.4991  | 0.5804  | Yes   |           |
|                 |              | Tiny-ImageNet | 0.0004  | 0.0059  | 1.2621  | 1.2777  | Yes   |           |
| **Tiny-ImageNet** |          |         |         |         |         |         |       | 100 (TPR) |
| Stolen          |              | Tiny-ImageNet | 0.0012  | 0.0053  | 0.3444  | 0.3444  | Yes   |           |
|                 |              | MNIST    | 0.0001  | 0.0126  | 0.4355  | 0.5156  | No    |           |

Figure 6. Performance of Meta-classifier for Tiny-ImageNet against MAA with DenseNet (DN), ResNet (RN), and VGG, DAA with two different versions which are from scratch (V1) and with pre-trained victim model (V2), MRA by dataset size, and MFA by number of data samples.

Figure 7. Performance of Meta-classifier for MNIST against MAA with DenseNet (DN), ResNet (RN), and VGG, DAA with two different versions which are from scratch (V1) and with pre-trained victim model (V2), MRA by dataset size, and MFA by number of data samples.

TASTER has met the four design requirements defined in Section 4. Firstly, to meet the robustness criteria, our DEEP TASTER demonstrated its ability to capture the dataset ownership IP and be resilient even to model architecture changes or changes in the number of model output classes as a form of attack. Secondly, to meet the fidelity criterior, our IP protection mechanism has zero impact on the model accuracy due to its design as a fingerprinting technique rather than the traditional watermarking invasive method. Thirdly, to meet the efficacy criteria, DEEP TASTER has exhibited high detection accuracy across six attacks with reliable TPR and TNR. Lastly, to meet the efficiency criteria, we conducted further experiments to investigate what is the minimum number of adversarial DFT samples used in the inference to detect an attack?

In this experiment, the CIFAR10 meta-classifier was used against 69 suspect models. For the positive cases, we use a combination of 37 models. This includes three CIFAR10 models with MPA attack, 12 CIFAR10 models with MFA attack, six CIFAR10 models with DAA attack (three with DAA scratch, three with DAA pretrained), three CIFAR10 models with TLA attack, and 12 CIFAR10 with MRA (70%, 90%, and 100% of dataset was used). For the negative cases, we use a combination of 32 models. This includes three ImageneNet benign models with different architecture, four MNIST benign models with various epochs, six MNIST benign models with DAA, 16 MNIST benign models with MFA, and three MNIST benign models with MPA. We found that with only three adversarial DFT samples, our sysname could detect the positive stolen from negative benign.

How does the meta-classifier threshold impact the DEEP TASTER model’s efficacy? As stated in Section 4.1.2, we define the threshold of the data classifier such that 4% of the samples in the victim dataset validation set lie above the threshold (or equivalently, 96% lie below). To determine this figure, we experimented with how the performance of three different meta-classifiers depends on the threshold. In the experiment, 9 models consisting of three datasets, three model architectures and three meta-classifiers targeting each dataset were used, and the performance index was set as balanced acuity. Figure 8 show the change in performance of each meta-classifier when the threshold value is changed from having the top 1% of the samples lying above the threshold to the top 10% of samples lying above the threshold. As
the threshold value decreases, the performance of the 3 classifiers tends to increase and then decrease. When calculated from the deviation values of the three meta-classifiers, the highest performance is 94.52% when the threshold is set to a value of top 4%. Accordingly, we conducted all our experiments by setting the threshold of all meta-classifiers as a value of top 4%. As our Meta-classifier is one class classifier, this threshold has been selected independently from the true-negative case.

**How the meta-classifier training dataset size and dimensions impact DEEPASTER model’s efficacy?** The performance of meta-classifier may depend on the size of training adversarial DFTs samples dataset. Generally, a larger training dataset might facilitate producing a higher performance model. In our case, generating large adversarial DFTs samples dataset might mean higher time cost. For generating balanced model between TPR and TNR, we test the relationship of performance and dataset size. Training dataset is generated with various size: 2400, 4800, 7200, 9600 images. These four dataset is generated from ImageNet VGG, ResNet, and Densenet models and we use the same set of adversarial DFTs sample for consistency. The BA of meta-classifier is 97.28%, 98.50%, 96.28%, and 96.82% when the training dataset size is 2400, 4800, 7200, 9600 respectively. We choose the training dataset size as 4800 as it produces the best performance.

We also observe that the performance of our DEEPASTER can vary depending on the adversarial image dimensions. The smaller the size of the adversarial image, the smaller the perturbation that could capture from the model dataset intelligence, the lower the performance of the DEEPASTER might be. If the size of the image is $32 \cdot 32 \cdot 3$, the model exhibits almost indistinguishable performance, but if the size of the image is $224 \cdot 224 \cdot 3$, as currently used in the experimental setting, the detection performance is high. Therefore, we recommend generating large-dimensional adversarial images when using DEEPASTER.

**Model IP vs. Data IP.** Existing IP verification approaches for DNN (e.g., [5], [8], [21], [44]) typically focus on a model’s explicit properties, such as the parameters or weights of a DNN model, namely the model IP, which are model dependent. Therefore, those existing DNN fingerprinting techniques would be ineffective in detecting data theft attacks. From the experimental results in the previous work [8] and our work (see Section 5.4), we confirmed that DEEPJUDGE recognized the cases in which different model architectures are built on the same dataset as separate models. In contrast, we consider a model’s implicit properties representing the knowledge learned from the training data, namely the data IP, which are the training dataset dependent. Instead of examining individual neuron-level metrics which are model dependent, we are trying to find the context features learned from adversarial examples in terms of the decision boundaries of a model, which would be more highly affected by the training dataset rather than the model itself. Even though we use different model architectures, we can obtain adversarial examples having statistically similar patterns in the DFT domain from each model architecture if we use the same dataset for training those models. Consequently, DEEPASTER can effectively detect data theft attacks even when attackers use a model architecture different from a victim’s.

**Adaptive Attacks.** In our experimental evaluations across the six targeted attacks, we extensively examined various adaptive attack strategies that could be employed by the attacker to evade dataset intelligence stealing detection. These include changing the model architecture in MAA, altering the number of classes in DAA and TLA, tuning the parameters in both MFA and MPA, and altering the proportion of the dataset used in the MRA. In all these cases, our DEEPASTER exhibits a robust ability to detect the stolen dataset IP, while being able to recognize the benign cases. The only exception is that when the attacker uses small proportion of the dataset in the retraining attack, our DEEPASTER might not be able to flag the new model as stolen. This is arguably an acceptable behavior as we are tracking the dataset intelligence, and stealing its intelligence would require the attacker to use large proportion of it, ≥ 70%, to be able to reproduce its utility.

We next discuss what adaptive strategy the attacker could employ once our defence mechanism is released? To mitigate that risk, our DEEPASTER relies on two key pillars which are the adversarial perturbation generation/ transformation and the meta-classifier. While we make no assumption around the adversarial generation, we assume that the meta-classifier pipeline including its parameters, training configurations and thresholds to be confidential and secure. It should be only accessible by the model owner or during the verification within a secure environment.

**Complexity Evaluation.** To calculate the complexity of DEEPASTER, we measure the time it takes to distinguish the suspect model using the meta-classifier. As summarised in Table 11, we have 3 steps. For step 1 - the Adversarial Perturbation Generation and Transformation - the time is around 769 seconds (0.3 seconds per image). For step 2 - training the meta-classifier - the time taken is around 766.8 seconds. Note, steps 1 and 2 are one-off tasks for model development and are not repeated for every verification. Only a few adversarial DFT samples are needed for verification. For step 3 - suspect model verification - is around 9.78
(0.0339 second). The total time for generation, training and verification is about 1,546.44. This is a reasonable time considering DEEPJUDGE takes a total of 1,937.79 seconds.

| Step | Task                          | Time (Sec)         |
|------|-------------------------------|--------------------|
| 1    | Adversarial DFT Generation    | 0.3538 (per image) |
| 2    | Meta-classifier Training      | 766.8              |
| 3    | Suspect Model Verification    | 0.0339 (per image) |

**TABLE 11. COMPLEXITY EVALUATION FOR DEEPTASTER.**

**Limitations and Future Work.** Despite the robust efficacy of DEEPJUDGE against the six attacks and existing works, we acknowledge the following limitations of our current design.

- Reference models for the meta-classifier. We found that the adversarial DFT images contain both the dataset and model architecture information, which are entangled together. The main focus of this work is to track the IP of datasets across architectures. We observe that if the suspect model architecture is previously unseen by the meta-classifier, it may result in a higher false positive rate for detection. Therefore, ideally, the dataset information could be separated from the architecture information, to achieve better detection performance for unseen models architectures. We currently intend to reduce the architecture impact by using a larger variety of architectures as the reference models (victim models) for training the meta-classifier. A future research direction may be to propose disentanglement strategies to separate the architecture information in the DFT images.

- White-box design. To evaluate the suspect model, DEEPTASTER needs access to the suspect model to be able to generate adversarial DFT samples against that model before examining them with our meta-classifier. A possible future direction could be to explore ways to generate adversarial samples in a black-box setting to provide more flexibility and generalization.

- Sensitivity of detecting retraining models. The detection performance of DEEPTASTER for the models retrained on 70% of the stolen dataset is robust, over 71.37% in general. However, when the percentage of stolen datasets drops to 30% or 20%, the detection accuracy is reduced to 1-2%. In the future, we may propose an adaptive threshold for increasing the sensitivity of the data IP detection performance.

**7. Related Work**

**7.1. DNN Watermarking**

The first stream of related work uses watermarking to protect the copyright of DNN models [2], [9], [15], [18], [34], [42]. As in classical multimedia watermarking, DNN watermarking includes two stages: embedding and verification. In the embedding stage, the DNN model owner inserts a secret watermark (e.g., signature or a trigger) into the model during the training phase. Existing watermarking techniques can be categorised as either white-box or black-box based on how much knowledge is available during the verification stage. White-box techniques assume the model parameters are available [9], [34], [37]. They insert a string of bits (signature) into the model parameter space via several regularization terms. The ownership of the IP could be claimed when the retrieved string of bits from the suspect model matches to the owner signature. Black-box techniques only have access to model predictions during verification. They leverage backdoor attacks [12] to embed a watermark (backdoor samples) into the ownership model during the training process, where the class of each backdoor sample is relabelled to a secret class [18], [42]. The ownership could be verified by querying the suspect model using the pre-defined backdoor samples and receiving the correct secret class for each sample.

**7.2. DNN Fingerprinting**

DNN fingerprinting mechanisms have been recently introduced as an alternative approach to verifying model ownership via two stages called fingerprint extraction and verification. Fingerprinting methods [4], [8], [21], [44] are all black-box techniques. They are non-invasive, as opposed to watermarking techniques that are invasive. Rather than altering the training process to inject the watermark, fingerprinting directly retrieves a unique property/feature of the owner’s model as its fingerprint. The ownership can then be validated if the fingerprint matches with the one extracted from the suspect model. In general, there are two streams of work under this category: single and multiple fingerprinting. Single fingerprinting uses one feature/property as identifier. For example, IPGuard [4] uses data points close to the model’s decision boundaries as that identifier. Lukas et. al. [21] propose a conferrable adversarial examples that transfers a target label from a source model to its stolen model. They use that as a model identifier. Multiple fingerprinting leverages multiple features/metrics as a fingerprint to handle different types of model stealing and adaptive attacks. For instance, DEEPJUDGE [8] recently introduced a multi-level metrics mechanism that could be used as a unique IP identifier between owner and stolen models.

Although the above streams protect the model IP with high performance, they suffer from two main limitations. Firstly, they are architecture-dependent by design. For example, training the same (stolen) dataset on 3 different DNNs cannot be identified as IP violation, even though all 3 models absorbed the same dataset ownership IP. Secondly, due to being architecture-dependent, they struggle to detect transfer learning attacks. For instance, if a pre-trained DNN is stolen and used for transfer learning to a different domain, this cannot be tracked as stolen IP. In other words, they could not track the dataset ownership IP obtained from a dataset across various architectures. Therefore, we propose DEEPTASTER, a robust dataset ownership IP tracking technique against 6 attacks.

**8. Conclusion**

In this paper, we proposed a novel fingerprinting technique dubbed DEEPTASTER which tracks the dataset IP using adversarial perturbations in the Fourier domain. We discovered that the learned knowledge of DNNs from a specific dataset can be exposed by the spectra of the gradient-based adversarial perturbation of the DNNs. That
DeepTASTER generates a few adversarial images using adversarial perturbations, and transforms them into the Fourier frequency domain before training a metaclassifier that can be used to verify whether a target dataset has been used in the training of a DNN model. To demonstrate the effectiveness of DeepTASTER we evaluated its detection accuracy on three datasets, with three model architectures, under various attack scenarios — including mutating the model architectures, transfer learning, pruning, fine-tuning, and data augmentation. Our results suggest that DeepTASTER is robust against all of these attacks.

References

[1] Alsharif Abuadbba, Hyoungshick Kim, and Surya Nepal. DeepSign: invisible fragile watermark to protect the integrity and authenticity of cnn. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, pages 952–959, 2021.

[2] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In 27th USENIX Security Symposium (USENIX Security 18), pages 1615–1631, 2018.

[3] William Aiken, Hyoungshick Kim, and Simon Woo. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks, 2020.

[4] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary, 2019.

[5] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, pages 14–25, 2021.

[6] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. Deepmarks: A digital fingerprinting framework for deep neural networks. arXiv preprint arXiv:1804.03646, 2018.

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

[8] Juiluo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. Copy, right? a testing framework for copyright protection of deep learning models. arXiv preprint arXiv:2112.05588, 2021.

[9] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pages 485–497, 2019.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009.

[11] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[12] Tianyu Gu, Kang Liu, Brendon Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access, 7:47230–47244, 2019.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[14] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[15] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In 30th USENIX Security Symposium (USENIX Security 21), pages 1937–1954, 2021.

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[17] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge.

[18] 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.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

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

[21] Nils Lukas, Yuxuan Zhang, and Florian Kirschbaum. Deep neural network fingerprinting by conferrable adversarial examples. arXiv preprint arXiv:1912.06888, 2019.

[22] Hengliang Luo, Yi Yang, Bei Tong, Fuchao Wu, and Bin Fan. Traffic sign recognition using a multi-task convolutional neural network. IEEE Transactions on Intelligent Transportation Systems, 19(4):1100–1111, 2017.

[23] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[24] Hannah Murphy and Shannon Bond. Capital One data breach sparks cloud security fears. The Financial Times, 2019.

[25] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, and Kurt M. Rieck. Deep k-classification. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 506–519, 2017.

[26] Jonas Rauber, Roland Zimmermann, Matthias Bethge, and Wieland Brendel. Foolbox native: Fast adversarial attacks to benchmark the robustness of machine learning models in pytorch, tensorflow, and jax. Journal of Open Source Software, 5(53):2607, 2020.

[27] Nils Lukas, Robert Vandermeulen, Nico Goernitz, Lucas Deleeke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 4393–4402. PMLR, 10–15 Jul 2018.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[29] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind signs: invisible fragile watermark to protect the integrity and authentication of cnn. In Proceedings of the 29th ACM International Conference on Multimedia, pages 242–264. IGI Global, 2010.

[30] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. In Proceedings of the 29th ACM International Conference on Multimedia, pages 4417–4425, 2021.

[31] David M. J. Tax and Robert P. W. Duin. Support vector data description. Machine Learning, 54:45–66, 2004.

[32] Lisa Torrey and Jude Shavlik. Transfer learning. In Handbook of Research on Machine Learning Applications and Trends, pages 242–264. IGI Global, 2010.

[33] Jean-Baptiste Truong, Pratyush Maini, Robert J. Walls, and Nicolas Papernot. Data-free model extraction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2021.

[34] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin’ichi Satoh. Embedding watermarks into deep neural networks. In Proceedings of the 2017 ACM on international conference on multimedia retrieval, pages 269–277, 2017.
[35] Bolun Wang, Yuanshun Yao, Shawn Shan, Huizing Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In 2019 IEEE Symposium on Security and Privacy (SP), pages 707–723. IEEE, 2019.

[36] Hao Wang, Yitong Wang, Zheng Zhou, Jing Ji, Dihong Gong, Jingchao Zhou, Zhipeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5265–5274, 2018.

[37] Shuo Wang, Sidharth Agarwal, Sharif Abuadbba, Kristen Moore, Surya Nepal, and Salil Kanhere. Integrity fingerprinting of dnn with double black-box design and verification. arXiv preprint arXiv:2203.10902, 2022.

[38] Shuo Wang, Surya Nepal, Kristen Moore, Marthie Grobler, Carsten Rudolph, and Alsharif Abuadbba. Octopus: Overcoming performance and privatization bottlenecks in distributed learning. IEEE Transactions on Parallel and Distributed Systems, 2022.

[39] Mingfu Xue, Jian Wang, and Weiqiang Liu. Dnn intellectual property protection: Taxonomy, attacks and evaluations (invited paper). In Proceedings of the 2021 on Great Lakes Symposium on VLSI, GLSVLSI ’21, page 455–460, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Yifan Yan, Xudong Pan, Yining Wang, Mi Zhang, and Min Yang. Cracking white-box dnn watermarks via invariant neuron transforms, 2022.

[41] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. Cloudleak: Large-scale deep learning models stealing through adversarial examples. In NDSS, 2020.

[42] 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 Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pages 159–172, 2018.

[43] Jianpeng Zhang, Yutong Xie, Qi Wu, and Yong Xia. Medical image classification using synergic deep learning. Medical image analysis, 54:10–19, 2019.

[44] Jingjing Zhao, Qingyue Hu, Gaoyang Liu, Xiaochong Ma, Fei Chen, and Mohammad Mehedi Hassan. Af3: Adversarial fingerprinting authentication for deep neural networks. Computer Communications, 150:488–497, 2020.