CHAPTER 20

Model watermarking and fingerprinting

Engineering a top-notch deep learning model is an expensive procedure, which involves collecting data, hiring human resources with expertise in machine learning, and providing high computational resources. For that reason, deep learning models are considered as valuable intellectual properties (IPs) of the model vendors. To ensure reliable commercialization of deep learning models, it is crucial to develop techniques to protect model vendors against IP infringements. One of such techniques, which recently has shown great promise, is *digital watermarking*.

Moreover, with the rapid development of machine learning and artificial intelligence, the efforts and resources spent in developing state-of-the-art machine learning models such as deep neural networks (DNNs) can be tremendous, and therefore it is of utmost importance to be able to claim the ownership of a well-trained model and its derived versions (e.g., pruned models). For instance, the cost of training current state-of-the-art transformer-based language model GPT-3 (Brown et al., 2020a) is estimated to be at least 4.6 million US dollars.[1] Imagine that an unethical model thief purposely pruned the pretrained GPT-3 model and attempted to claim the ownership of the resulting compressed model. The solution to the challenge of “how to protect IP for DNN models and reliably identify model ownership?” is literally worth million dollars. The methods for model watermarking and fingerprinting, as introduced in this chapter, are motivated from the study of adversarial robustness for DNNs.

20.1 Model watermarking

Aramoon et al. (2021) present GradSigns, a novel watermarking framework for deep neural networks (DNNs). GradSigns embeds the owner’s signature into the gradient of the cross-entropy cost function with respect to inputs to the model. Their approach has a negligible impact on the performance of the protected model, and it allows model vendors to remotely verify the

[1] https://bdtechtalks.com/2020/08/17/openai-gpt-3-commercial-ai.

Copyright © 2023 Elsevier Inc.
All rights reserved.

watermark through prediction APIs. Table 20.1 lists the properties that an effective watermarking technique for deep learning models should have.

Table 20.1 Properties of an effective watermarking technique for deep learning models.

Properties	Description
Loyalty	Watermark should have negligible overhead on the model's performance.
Robustness	Watermark must remain verifiable in the presence of antiwatermark attacks.
Reliability	Watermark verification should result in minimal false ownership claims.
Credibility	Finding or forging a fake (ghost) watermark should not be feasible.
Efficiency	Watermark extraction and verification should incur low costs.
Capacity	Watermarking technique should be able to embed large signatures.

Threat model. The threat model includes two parties, *model vendor* and *adversary*. The model vendor owns model M, a DNN that they have engineered and trained for a certain task T using the dataset D. The dataset D is collected and owned by the model vendor. The second party, the adversary, is an entity that does not have the required resources for designing and training a top-notch model and wishes to make a profit out of model M without paying any copyright fee to the model vendor. The adversary can be a company that has purchased the license of M for one of their products and wants to deploy it on another one without paying additional copyright fees. They can also be any entity that has somehow got their hands on the model and wishes to sell it on the darknet. Model vendor's goal is to protect M against IP infringements by means that enables the vendors to prove their ownership and possibly detect the source of theft. On the other hand, the adversary's ultimate goal is to continue profiting from M without getting caught by law enforcement.

The threat model in (Aramoon et al., 2021) assumes the strongest adversary who has the expertise and the computation power required for training a model. However, the dataset that they have available for task T is far smaller than dataset D owned by the model vendor and therefore is not large enough for them to train a top-grade model from scratch. If the adversary had access to dataset D, then they would not need to hijack M as they are capable of training the model themselves. Similarly to prior arts, it is assumed that the adversary is capable of trying any of the general antiwatermark attacks such as parameter pruning, model finetuning, and query invalidation and modification to remove the watermark or ob-

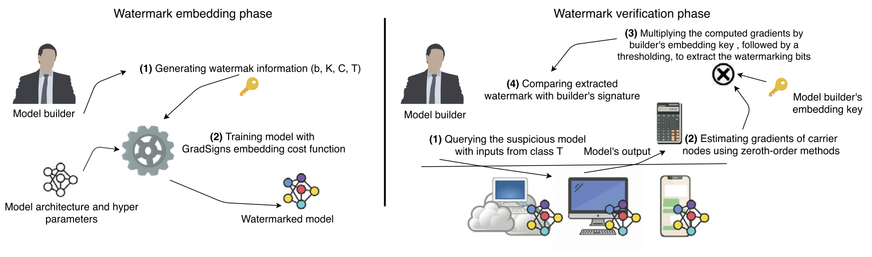
Watermark embedding phase
Model builder
(1) Generating watermak information (b, K, C, T)
(2) Training model with GradSigns embedding cost function
Model architecture and hyper parameters
Watermarked model
Watermark verification phase
(3) Multiplying the computed gradients by builder's embedding key , followed by a thresholding, to extract the watermarking bits
(4) Comparing extracted watermark with builder's signature
Model builder
Model builder's embedding key
(1) Querying the suspicious model with inputs from class T
Model's output
(2) Estimating gradients of carrier nodes using zeroth-order methods

Figure 20.1 Workflow of watermark embedding and verification using GradSigns proposed in (Aramoon et al., 2021).

struct verification. In addition, it is assumed that the adversary is aware of all existing watermarking techniques and is capable of designing adaptive antiwatermark schemes to target the deployed method. Due to such possible counter-watermark attempts, it is safe to assume that a hijacked model will go under modifications before being monetized by the adversary. The adversary accomplishes their goal if he can remove the vendor's signature or obstruct watermark verification without sacrificing too much on the performance of the model. Note that a counter-watermark attempt that drastically degrades the model's performance is not considered successful.

Methodology of GradSigns. Intuitively, GradSigns finds a solution, i.e., a set of model parameters, corresponding to a decision boundary that not only results in comparable performance on the original task but also fulfills an additional goal carrying the owner's watermark information. GradSigns works by embedding watermark information into the expected gradient of the cross-entropy cost function with respect to the model's input. For any input sample x, the gradient of the cost function with respect to the input is a vector tangent to the model's cost function surface and perpendicular to the decision boundary at point x. Therefore, by imposing a statistical bias on these gradients, GradSigns is essentially reshaping the decision boundary to incorporate the desired watermark information. For simplicity, we refer to the gradient of the cross-entropy cost function with respect to the input of the model as the *gradient of input* or *input gradient*. Fig. 20.1 illustrates the workflow of GradSigns.

Watermark embedding. Forcing a random statistical bias on the gradient of inputs to the model can drastically degrade its performance on the classification task. To ensure a successful marking without sacrificing the model's performance, the watermark needs to be embedded while optimizing the model for the original task. For that reason, Aramoon et al. (2021) embed the watermark into the host model by including a regularizer term in the model's training cost function. The final training cost function including the regularizer term is defined as

$$J(x|\theta, y) = J_{\text{cross-entropy}}(x|\theta, y) + \lambda J_{\text{embedding}}(x|\theta, y), \tag{20.1}$$

where θ denotes the model's parameters, x is an input sample, y is the ground truth label for input sample x, $J_{\text{cross-entropy}}$ is the task-specific cost function, which is the cross-entropy function for classification problems, λ is the trade-off hyperparameter, and $J_{\text{embedding}}$ is the watermark embedding regularizer term, which penalizes the distance between the expected value of input gradients and the desired watermark. Before defining the embedding regularizer term, we explain the steps that the model vendor needs to take prior to embedding the watermark. These steps are as follows:
Step 1. Generating an N-bit vector $b \in \{0, 1\}^N$ to be used as the watermark.
Step 2. Randomly selecting a set C of input neurons to carry the watermark. We refer to set C as the *watermark carrier set*, and to neurons in C as *carrier nodes*. The gradient of inputs observed on neurons in the career set participates in embedding the watermark.
Step 3. Generating an *embedding key* $K^{N\times|C|} \in [-1, 1]^{N\times|C|}$. An embedding key is a transformation matrix that maps the expected gradient of carrier nodes to a binary vector of size N.
Step 4. Selecting a random target class T. Images from class T are used to calculate the gradients of carrier nodes.

Note that generating the watermark b and embedding key K can either be done randomly by using a random number generator (RNG) or by hashing a message containing information that can be used to prove vendor's ownership. In GradSigns the watermark is successfully embedded if the following property holds:

$$\forall j \in \{0, 1, \ldots, N-1\}, \ \ \chi_{[0,\infty)}\left(\sum_{i=0}^{|C|-1} K_{ji} G_i\right) = b_j, \tag{20.2}$$

where $G \in \mathbb{R}^{|C|}$ is the expected gradient of cross-entropy function with respect to carrier nodes in C, measured over a sample of images from target class T, K is the model vendor's embedding key, b_j is jth watermark bit, and χ is a step function outputting one for values greater than zero.

For each watermark bit j, (20.2) denotes a linear inequality where the expected gradients of carrier nodes (G) are the variables, and the jth row of the embedding key K is the coefficients, as shown in (20.3). This linear inequality is essentially denoting a half-space where acceptable values of expected gradients can reside for a successful embedding of watermark bit j:

$$(-1)^{b_j} \sum_{i=0}^{|C|-1} K_{ji} G_i < 0. \tag{20.3}$$

The task of embedding each watermark bit j can also be viewed as a binary classification task with a single-layer perceptron (SLP), where the parameters of the perceptron layer are fixed to a constant value equal to the jth row of the embedding key K, and only the input of the SLP, i.e., the gradient of the carrier nodes, is being trained. To this end, we use a binary cross-entropy loss function in the embedding regularizer to embed each watermark bit:

$$J_{\text{embedding}}(\theta) = -\sum_{j=0}^{N-1} (b_j \log(y_j) + (1 - b_j) \log(1 - y_j)), \tag{20.4}$$

where $y_j = \sigma(\sum_i K_{ji} G_i)$ is the output of the SLP corresponding to the jth watermarking bit, and σ is the sigmoid function.

Watermark extraction. To extract a watermark embedded by GradSigns, the first step that the model owner needs to take is computing the expected gradient of the carrier nodes. In the white-box setting, where the owner has access to the internal configurations of the suspicious model, the gradients can be calculated by backpropagation. However, in the black-box setting, computing gradients via backpropagation is not possible.

To enable watermark extraction in the black-box setting, the zeroth-order gradient estimation method (see Chapter 3) is used to calculate the expected gradients of the carrier nodes. Zeroth-order methods can estimate gradient with respect to any direction v by evaluating the cost function value at two very close points located along this direction (Ghadimi and Lan, 2013; Liu et al., 2020a). The difference quotient is used to estimate the gradient of cost function with respect to carrier nodes as follows:

$$\widehat{G_c}(x) = \frac{\partial J(x)}{\partial x_c} \approx \frac{J(x + he_c) - J(x)}{h}, \tag{20.5}$$

where $\widehat{G_c}(x)$ is the estimated gradient of carrier node c at point x, h is the estimation step length, e_c is a standard basis vector with 1 at the component corresponding to career node c and 0s elsewhere. Note that the value of cross-entropy function $J(x)$ can be computed in the black-box setting, given model's output and the ground truth label for input x.

In (20.5), for each input x, the gradient of a carrier node is calculated by evaluating the value of the cost function for two points whose coordinates are the same except for the one coordinate corresponding to the carrier node. The gradient estimation error of career nodes, not including the error introduced by limited numerical precision, is of order $O(|C|h^2)$ (Liu et al., 2018b). For any input x, we need to evaluate the cost function $|C| + 1$ times to estimate the gradients of all carrier nodes. Note that the watermark extraction naturally applies to more query-efficient gradient estimation methods such as (Liu et al., 2019c). After calculating the expected gradients for all carrier nodes, the model vendor can retrieve the embedded watermark by multiplying expected gradients with their embedding key. The model belongs to the vendor if the bit error rate (BER) of the extracted watermark is lower than a certain threshold.

We refer the readers to the detailed experiments and analysis in (Aramoon et al., 2021) for property evaluation of watermarking techniques with respect to Table 20.1 and for robustness assessment against counter-watermark attacks.

20.2 Model fingerprinting

Many existing DNN IP protection methods require intervention in training phase, which may cause performance degradation of the DNN (i.e., accuracy drop). Moreover, existing works may overlook the false positive problem of the DNN (i.e., mistakenly claiming the ownership of irrelevant models), which is of practical importance when designing fingerprints.

To address these limitations, Wang et al. (2021d) propose a novel approach to fingerprinting neural networks using *characteristic examples* (C-examples). Its advantages are as follows: (i) its generation process does not intervene with the training phase, and (ii) it does not require any realistic data from the training/testing set. By applying uniform random noise to the weights of the neural network with the combination of gradient mean descending technique, the proposed C-examples achieve high-robustness to the resulting models pruned from the base model where the fingerprints are extracted. When further equipped with a high-pass filter in the frequency domain of image data during the generation process, C-examples attain low-transferability to other models different from the base model. The C-examples are significantly different from widely known adversarial examples

(Szegedy et al., 2014; Goodfellow et al., 2015) causing model misprediction. Instead, C-examples are data-free and aim to achieve *high-robustness* for passing through pruned variants of the base model and *low-transferability* for screening out any other models different from the base model.

Wang et al. (2021d) consider three types of DNN models that are of interest in C-examples. ① *Base Model*: the pretrained model to fulfill some designated task, such as image classification. ② *Pruned Models*: the models pruned from the base model and implemented on the edge devices for inference execution. ③ *Other Models*: any other models that are neither ① nor ②. For example, if VGG16 is the base model, then VGG19, the ResNet family, etc. all belong to other models. The proposed DNN fingerprinting framework is as follows: The ① base model is used to generate C-examples with labels. Therefore C-examples have 100% accuracy on the base model. Then C-examples are used as fingerprints to test the models implemented on edge devices. A high accuracy is expected if the implemented model is ②, whereas a low accuracy is expected if the implemented model is ③. A systematic illustration of the C-example generation process is shown in Fig. 20.2.

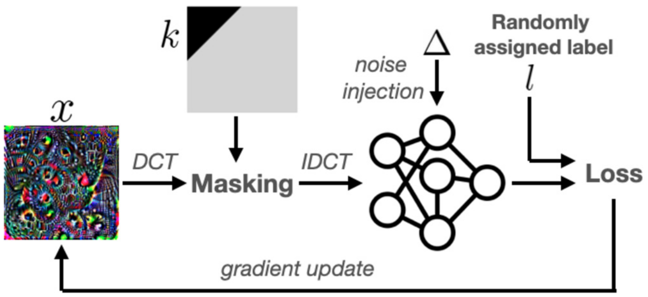

Figure 20.2 A system diagram of generating LTRC-example proposed by Wang et al. (2021d).

Vanilla C-examples. Let $\mathbf{x} \in \mathbb{R}^{3\times H\times W}$ denote a colored RGB image, where H and W are the image height and width, respectively. We scale pixel values of $\mathbf{x}$ to $[0, 1]$ for mathematical simplicity. F_θ denotes the base model, which outputs $\boldsymbol{y} = F_\theta(\mathbf{x})$ as a probability distribution for a total of K classes. The element y_i represents the probability that an input $\mathbf{x}$ belongs to the ith class. The base model F_θ parameterized with θ is pretrained. Then C-examples are generated from F_θ. If we are given with a subset $\{l_1, l_2, \dots, l_P\}$ of P labels randomly chosen from the labels of training dataset, then a set of η-optimal C-examples X^* can be characterized as

$$X^* = \left\{(\mathbf{x}, l) \middle| \text{Loss}_\theta(\mathbf{x}, l) < \eta, \mathbf{x} \in [0, 1]^n\right\}. \tag{20.6}$$

The term Loss_θ denotes the loss function of F_θ. A C-example $\mathbf{x}$ minimizing the loss for a specified label l should satisfy the above constraint. To be independent of data when extracting the base model features, we simply use a random seed to generate a C-example, and therefore the generated C-examples are distinct from natural images for the human perception. A vanilla version C-example is shown in Fig. 20.3(a).

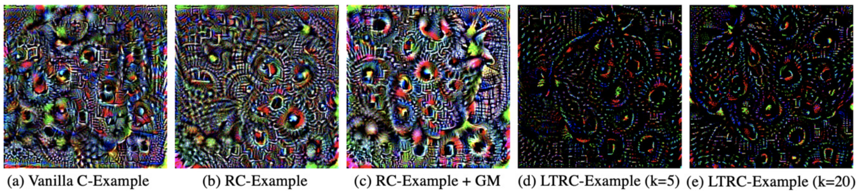

Figure 20.3 Characteristic examples visualized using different generation processes. The label assigned to all these image is "strawberry".

We can use the projected gradient descent (PGD) algorithm as introduced in Chapter 3 to find C-examples. The C-example generation problem (20.6) can be solved with the PGD algorithm as follows:

$$\mathbf{x}^{t+1} = \mathbf{Clip}\left(\mathbf{x}^t - \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} \text{Loss}_\theta(\mathbf{x}^t, l))\right), \tag{20.7}$$

where t is the iteration step index, $\mathbf{x}^0$ is the random starting point, α is the step size, sign returns the elementwise sign of a vector, $\nabla_{\mathbf{x}}$ calculates input gradients, and **Clip** denotes the clipping operation to satisfy the $\mathbf{x} \in [0, 1]^n$ constraint.

Robust C-examples. Wang et al. (2021d) further propose an enhancement named *Robust C-examples (RC-examples)* over the vanilla C-examples, by adding noise bounded by ϵ to the neural network parameters θ to mimic the model weight perturbation due to the model compression procedure. Here the loss is changed to $\text{Loss}_{\theta+\Delta}$, where Δ presents the uniformly distributed weight perturbations within $[-\epsilon, \epsilon]$. Moreover, motivated by the expectation-over-transformation (EOT) method, (Athalye et al., 2018) toward stronger adversarial attacks (see Chapter 4 for details for EOT attack), the proposed RC-examples can be further enhanced by calculating the mean of the input gradients (the gradient mean (GM) method) in each iteration step over different random realizations of Δ.

LTRC-examples. Finally, in addition to enhancing the robustness of C-examples on ② Pruned Models, it is desirable to exhibit low-transferability to ③ Other Models. Therefore the RC-examples are further improved by enforcing low transferability, named the *low-transferability RC-examples (LTRC-examples).* In this way, we can improve the capability of C-examples in detection for *false positive* cases, where positive means claiming the model ownership as ours in IP protection.

The frequency analysis (Guo et al., 2018; Sharma et al., 2019; Cheng et al., 2019b) suggests that low-frequency components can improve transferability of adversarial examples. Inspired by that, we propose to leverage high-frequency components to achieve C-examples with low transferability. Specifically, we can apply a frequency mask on the *discrete cosine transform* (DCT) (Rao and Yip, 2014) to implement a high-pass filter in the frequency domain of the C-example. As an important tool in signal processing, the DCT decomposes a given signal into cosine functions oscillating at different frequencies and amplitudes. For a 2D image, the DCT performed as $\omega = \mathrm{DCT}(\mathbf{x})$ can transform the image $\mathbf{x}$ into the frequency domain, and $\omega_{(i,j)}$ is the magnitude of its corresponding cosine functions with the values of i and j representing frequencies, where smaller values mean lower frequencies. The DCT is invertible, and the inverse DCT (IDCT) is denoted as $\mathbf{x} = \mathrm{IDCT}(\omega)$. Note that Wang et al. (2021d) apply DCT and IDCT for different color channels independently.

Inspired by the observation that the low frequencies play a more important role in machine classification and therefore are more transferable, Wang et al. (2021d) propose to filter out these components to effectively lower the fingerprint transferability. To demonstrate this, a high-pass frequency mask shown in Fig. 20.2 is imposed, where the high-frequency band size k controls the range of the filtered low-frequency components. The frequency mask is designed to be a 2D matrix with elements being either 0 or 1, i.e., $\mathbf{m} \in \{0,1\}^{H\times W}$, which performs elementwise product with the DCT of C-example. At each iteration step to generate the fingerprints, the high-pass mask sets the low-frequency components to 0, i.e., $\omega_{(i,j)} = 0$ if $1 \le i + j \le k$, while keeping the rest of the high-frequency components. By using the high-pass frequency mask the LTRC-example at the $(t+1)$th iteration step can be derived by

$$\mathbf{x}^{t+1} = \mathrm{HighPass}\Big\{\mathbf{Clip}\left(\mathbf{x}^{\mathrm{t}} - \alpha \cdot \mathrm{sign}(\nabla_{\mathbf{x}}\mathrm{Loss}_{\theta+\Delta}(\mathbf{x}^{\mathrm{t}}, l))\right)\Big\}, \tag{20.8}$$

where the HighPass filter is defined as

$$\text{HighPass}(\cdot) = \text{IDCT}(\text{FrequencyMask}(\text{DCT}(\cdot))). \tag{20.9}$$

20.3 Empirical comparison

Here we compare different variants of the C-examples proposed by Wang et al. (2021d). Their visual comparisons are shown in Fig. 20.3.

In the experiment the accuracy of the C-examples on the pruned model is used to indicate its robustness and the accuracy on the variant model (with similar functionality to the base model, e.g., VGG-19 model to the base VGG-16 model) to indicate its transferability. Originally, the accuracy of all kinds of C-examples on the base model is 100% during generation. To effectively evaluate the trade-off between robustness of the pruned models and transferability to other variant models, we define the difference between the robustness and transferability as the *uniqueness score* (*uniqueness score* $=$ *robustness* $-$ *transferability*), where higher uniqueness score means that the C-examples are more robust to pruned models and less transferable to variant models. Intuitively, a better fingerprint method should achieve higher uniqueness score. The uniqueness score can also be used to indicate the false positive problem, i.e., if the uniqueness score is negative, then the corresponding fingerprint method is prone to make false model claims.

Table 20.2 demonstrates the effectiveness of C-examples, RC-examples, and LTRC-examples on different pruned models using the base VGG-16 model on ImageNet dataset with different pruning ratios for evaluating robustness. For testing transferability to other variant models (such as VGG-19, ResNet Family, DenseNet Family), as VGG-19 is the most similar architecture to VGG-16 and more transferable for fingerprints generated on VGG-16, we only report the transferability on VGG-19 and omit the transferability results on other models such as ResNet Family or DenseNet Family. Note that the transferability to other models should be lower than VGG-19, leading to better performance with higher uniqueness score.

As shown in Table 20.2, the uniqueness scores of RC-examples, RC-examples+GM, and LTRC-examples are higher than that of the baseline vanilla C-examples. We notice that C-examples suffer from negative uniqueness scores due to their high transferability to other models when the pruning ratios are 70% and 80%. We can observe that LTRC-examples with $\epsilon = 0.001$ achieve the best uniqueness scores with relatively large margins

Table 20.2 Uniqueness Score of C-examples on implemented models by different weight pruning on the base VGG-16 model with ImageNet dataset: The base model has 70.85% top-1 accuracy and 90.10% top-5 accuracy. The base model is pruned by unstructured pruning (Han et al., 2015) with various pruning ratio, where it is pruned for 5 times at each pruning ratio with average accuracy degradation for pruning ratio 40%, 50%, 60%, 70%, and 80% are 0.26%, 0.45%, 0.38%, 0.61%, and 0.97%, respectively. The LTRC-examples are set with $k = 20$. The robustness at each pruning ratio can be obtained by the summation of *Uniqueness Score* and transferability. The experiment is evaluated on 100 C-examples generated from VGG-16.

Method	ϵ	**Base Model VGG-16 (%)**	**Transferability to VGG-19 (%)**	**Uniqueness Score (%)**				
				40% Pruned	**50% Pruned**	**60% Pruned**	**70% Pruned**	**80% Pruned**
Vanilla C-Example	0	100	47	+13	+13	+13	−5	−22
RC-Example	0.001	100	60	+30	+30	+23	+22	+0
	0.003	100	88	+6	+11	+5	+2	−3
	0.005	100	86	+12	+12	+12	+9	+2
	0.007	100	95	+4	+4	+3	+2	+3
RC-Example+GM	0.001	100	55	+34	+37	+35	+23	−7
	0.003	100	67	+30	+38	+38	+21	+19
	0.005	100	85	+15	+15	+15	+15	+15
	0.007	100	100	+0	+0	+0	+0	+0
LTRC-Example	0	100	16	+53	+51	+50	+23	+13
	0.001	100	24	**+65**	**+65**	**+65**	**+58**	**+32**
	0.003	100	75	+25	+25	+25	+25	+23
	0.005	100	96	+4	+4	+4	+4	+3
	0.007	100	98	+2	+2	+2	+2	+2

Table 20.3 False alarm analysis of C-examples using AUC and F1-score. $k = 20$ is used for LTRC-example.

Method	ϵ	AUC	F1-score
Vanilla C-Example	0	0.87	0.91
RC-Example	0.001	0.94	0.91
	0.003	0.96	0.91
	0.005	0.98	0.95
	0.007	0.97	0.95
RC-Example +GM	0.001	0.97	0.95
	0.003	0.99	0.95
	0.005	0.99	0.95
	0.007	0.86	0.95
LTRC-Example	0.001	**1**	**1**
	0.003	**1**	**1**
	0.005	**1**	**1**
	0.007	**1**	**1**

(about 1.9×, 2.1×, and 5× that of the RC-examples+GM, RC-examples, and C-examples). In general, for a given method with fixed ϵ, the uniqueness score decreases if the pruning ratio increases since larger pruning ratio degrades the test accuracy, leading to weaker model functionalities with less robustness after pruning. Meanwhile, we observe that with increasing ϵ, there is more uncertainty in the model with larger random perturbations, leading to more general C-examples to incorporate larger uncertainty. Therefore they become more transferable to other variant models, resulting in increasing transferability and decreasing uniqueness score.

False alarm and utility analysis. Wang et al. (2021d) also evaluate the proposed C-examples under the false alarm scenario. Given a group of legally pruned models (test accuracy drop $< 2\%$) and other widely used variant models on ImageNet dataset including VGG19, ResNet50, ResNet101, ResNet152, DenseNet121, DenseNet169, and DenseNet201, they evaluate the effectiveness of C-examples by calculating the receiver operating characteristic (ROC) curve of each method with different ϵ and report the area under the curve (AUC) and F1-score corresponding to each method, shown in Table 20.3. For the pruned models, five pruned models corresponding to pruning ratios of 40%, 50%, 60%, 70%, and 80% are used for testing. We can observe that with LTRC-examples, the AUC and F1-score both reached the ideal case of 1 with all ϵ values, meaning that with an appropriate threshold, LTRC-examples as fingerprints will not cause the false alarm problem (i.e., recognize other variant models as the base model). Meanwhile, we notice that incorporating enhanced robustness (RC-examples vs. C-examples), GM (RC-examples+GM vs. RC-examples) or low-transferability (LTRC-examples vs. RC-examples+GM) can help with false detection issues and improve the AUC and F1-score.

20.4 Extended reading

- Shao et al. (2021b) use adversarial examples to design robust text CAPTCHA (Completely Automated Public Truing test to tell Computers and Humans Apart), which is a widely used technology to distinguish real users and automated users such as bots.
- Shan et al. (2020) use data perturbations to improve the data privacy of facial recognition models.
- Sablayrolles et al. (2020) use data perturbations to detect whether a dataset is used for training or not.