Staining and locking computer vision models without retraining

Oliver J. Sutton*
Synoptix Ltd. *and*
King's College London
oliver.sutton@kcl.ac.uk

Qinghua Zhou*
King's College London
qinghua.zhou@kcl.ac.uk

George Leete
Synoptix Ltd.

Alexander N. Gorban
Central Univ. Moscow

Ivan Y. Tyukin
King's College London *and* AI Center, Bol'shoi Bul'var 30, Moscow
and ISP RAS Research Center for Trusted Artificial Intelligence

Abstract

We introduce new methods of staining and locking computer vision models, to protect their owners' intellectual property. Staining, also known as watermarking, embeds secret behaviour into a model which can later be used to identify it, while locking aims to make a model unusable unless a secret trigger is inserted into input images. Unlike existing methods, our algorithms can be used to stain and lock pre-trained models without requiring fine-tuning or retraining, and come with provable, computable guarantees bounding their worst-case false positive rates. The stain and lock are implemented by directly modifying a small number of the model's weights and have minimal impact on the (unlocked) model's performance. Locked models are unlocked by inserting a small 'trigger patch' into the corner of the input image. We present experimental results showing the efficacy of our methods and demonstrating their practical performance on a variety of computer vision models.

1. Introduction

Designing, training and validating a computer vision model can be an expensive undertaking. The effect on an organisation if such a valuable investment is leaked or stolen, can be deleterious, particularly since model weights can be difficult for a human judge to distinguish confidently. Developing methods for protecting such intellectual property, which may be deployed on a vulnerable edge device, is therefore crucial.

Two natural approaches that a model's owner may wish to take are either to imprint an identifiable *stain* within its weights, or to add a *lock* to make the model unusable without adding a separate secret unlocking trigger to the model's input. Staining, also known as watermarking in the literature [5, 23], can be viewed as a form of 'copyright trap'. By embedding specific (unnatural) behavioural quirks into the model, an identifying fingerprint is produced which can later be used to determine the provenance of a stolen or copied model. Locking, on the other hand, has a more direct aim [6, 14, 16]: unless the thief knows the secret trigger, they are unable to use the model to its full capability.

Here, we present new methods for both staining and locking neural network models in computer vision. Unlike existing approaches in the literature (reviewed in Sec. 2), our algorithms do not require model training or fine-tuning. They directly modify the model's weights, and can therefore be used to stain or lock pre-trained models with minimal performance impact. Since re-training is not required, a single model can be supported and updated across multiple customers, while providing each with a separate stain or lock. A further key advantage of our staining and locking techniques is that implementing them does not require access to training or validation data. We also provide two distinct novel theoretical guarantees on the worst-case false positive rate of our staining and locking algorithms, analogues of which do not appear to be present in the literature.

Staining methods are broadly composed of two parts: a *schema* describing the information encoded in the stain and how it may later be read, and a *mechanism* through which this information is implanted into the model. Numerous schemas have been proposed in the literature, *e.g.* hiding a message in the model's weights, its activations or its class predictions for specific trigger inputs. Until now, however, only two families of mechanisms have been available, both requiring model retraining: through special loss functions [7, 8, 11, 28, 45, 46], or by manipulating the training data [1, 17, 26, 32]. Our novel training-free staining mechanism implants highly selective *detector neurons* into the model, which produce a strong output only when a special *trigger* is present in the model's input. Sec. 3.1 shows how several standard staining

*These authors contributed equally and are listed alphabetically.

schemas may be implemented using this mechanism.

Our lock may be seen as an 'active' variant of the stain mechanism. Additional *disruptors* are also implanted to pollute the model's activations and prevent it from performing well. A specially constructed 'trigger patch' is added to one corner of the input image during inference is detected by the detector neuron. The detector neuron then produces a signal which deactivates the disruptors, recovering the full model performance. Examples of such trigger patches are shown in Appendix B. Our locking methods are presented with a focus on their fundamental building blocks, which can then be used to design bespoke algorithms to lock specific models.

Our goal is to present these new techniques as clearly as possible in a simple setting. For this, we focus primarily on convolutional models, although extensions of the core algorithms to generative adversarial networks (GANs) and vision transformers (ViTs) are presented in Appendix E. Our staining methods are generic and apply directly to many different model architectures. Our basic lock is more architecture-dependent since it interacts with the fundamental action of the model. To provide architectural-independence, we also show how the lock may be embedded within a Squeeze-and-Excite block [21], a common component of many models. We demonstrate, in Sec. 6, that such a block can be added to many pre-trained models with minimal impact on performance. For simplicity and brevity, we do not focus too strongly on techniques for hiding and obfuscating the stains and locks; we provide ideas for achieving this throughout, but do not pursue it in detail to present the core concepts as clearly as possible in the space available. For practical success, we also feel it is preferable for a diversity of obfuscation techniques and algorithmic variations to proliferate. Instead, we focus on the simplest case where a single detector and disruptor are added to the model; in practice, multiple detectors should be combined to deactivate multiple disruptors. This straightforward extension of the methods presented here makes reverse-engineering the lock a significant challenge.

Our approaches to staining and locking are inspired by *stealth attacks* [41, 43, 44], which have recently been shown to enable an attacker to specify the output of models from many modern families for a given piece of input data by directly modifying just a few weights. By harnessing this core sensitivity of modern models, and exploiting the fundamental concentration properties of their feature spaces, the highly selective detector neuron of our stain and lock can be implanted without retraining the model. This detector forms the heart of our methods, around which completely new mechanisms are built to facilitate staining and locking. This link means that the novel theoretical guarantees we present in Sec. 5 can be adapted back to study stealth attacks, and complement those available in the literature [41, 43, 44].

The paper is organised as follows. Sec. 2 places our approach in the context of related work. Our staining and locking algorithms are presented in Sec. 3 and Sec. 4. Sec. 5 presents novel theoretical guarantees on the detector's false positive rate. Experimental results demonstrate the practical performance of our algorithms in Sec. 6. A discussion of our findings is in Sec. 7, and some final conclusions are given in Sec. 8. Our mathematical notation is summarised in Appendix A (in the supplementary material).

2. Related work

Staining. Several staining methods have been proposed in the literature and mainly fall within two thematic groups; see [5, 23] for a thorough review. *Loss-based methods* encode a secret message into the model's weights during training using a special loss function. This message may then be revealed using a special trigger input to the network. Common approaches encode the message as a binary string in the model's weights [7, 45], activations [28] or outputs [8, 11], either directly or to be decoded by a special model [10, 46]. *Backdoor methods*, on the other hand, manipulate training data to produce a unique behavioural signature. One method is to force the model to learn specific non-natural labels for certain distinctive inputs [1, 15, 25, 32]. Alternatively, other work has considered the task of training the model to learn to associate a specific class with certain deliberate changes to natural images [17], such as the presence of a logo [26].

Locking. Two key families of approaches have been proposed in the literature. *Intrinsic methods* embed the lock into the model itself during training. This may be by training the model's output to depend on an additional (secret) input [14, 42], a cryptographic scheme [33], a secret global mask applied to inputs [6], or by converting training and inference images using a 'style-transfer' model (*e.g.* to oil paintings) [16]. *Extrinsic methods* offer additional orthogonal security to intrinsic methods via techniques from outside the neural network domain, such as encryption [2] or secure infrastructure [35, 40].

Novelty. In contrast to those in the literature, our staining and locking algorithms directly modify a model's weights, so require no model training, fine-tuning or external infrastructure. This is a key advantage because a single base model can be supplied to many clients with different locks/stains, without re-training each time. Moreover, re-training a model carries the inherent risk of changing its behaviour in unidentifiable ways (especially if the model is tuned to give incorrect responses to natural images as in backdoor methods), which is avoided through our direct approach. Being embedded in the model itself, a lock also remains active even if an adversary gains access to the hardware on which it runs. Our provable performance guarantees further differentiate our methods; comparable results are not available for any of the methods in the literature.

3. Staining

This section presents the building blocks of our novel training-free staining mechanism. The stain implants a highly selective *detector neuron* into the model, which is activated by a specially optimised trigger input to the model. We show how this mechanism can be used to implement standard staining schemas in Sec. 3.1.

Multilayer perceptrons. Algorithm 1 outlines our mechanism for staining a generic multilayer perceptron neural network. A detector neuron with a weight vector sampled from $\mathcal{U}(\mathbb{S}^{m-1})$ (*i.e.* uniformly from the surface of a sphere) is inserted into the model. Concentration properties in the model's feature space ensure that this randomly sampled neuron is very unlikely to respond strongly to natural inputs (see Sec. 5). It is possible, however, to optimise a trigger input (*e.g.* using gradient descent) to strongly activate the detector. The strong response to the trigger input, therefore, becomes an identifying characteristic of the model and can be used as the basis for implementing standard staining schemas. To ensure that the detector neuron has no impact on the model's performance, the parameters δ and Δ can be used in conjunction with the activation function to ensure that the detector's response is zero for natural inputs.

The optimisation of the trigger input x^* is important here: a strong detector response to x^* is very distinct from its response to natural inputs, and the guarantees on the false positive rate are improved (see Theorem 1). It also means the response Δ can be achieved by scaling down the weight vector, which can be beneficial for evading detection.

Additive staining. Alternatively, an *additive stain* adds the detector weight to an existing neuron's weight. Once again, the detector component of the weight likely only responds weakly to natural inputs, meaning that it has little impact on the model's operation. The strong response to the trigger input, however, is still preserved. Since this method does not produce a 'silent' detector neuron, it is much less likely that the detector will be identified and removed by an adversary.

Convolutional networks. Algorithm 2 demonstrates how this framework applies to a convolutional neural network. For simplicity, this algorithm focuses on the non-additive stain, but also applies to models incorporating batch normalisation. A single convolutional kernel is used as the detector, and a reduction operator r is used for the optimisation to convert the kernel's activation map into a single scalar response value. This could, for example, be the response of the kernel at a particular position in the image, or its average response. Larger kernels are helpful in reducing the false positive rate, as made explicit by Theorem 1: because a $\kappa \times \kappa$ convolutional kernel with c input channels has $\kappa^2 c$ parameters, large kernels operate in a higher dimensional setting.

Extensions. Extensions to apply these staining algorithms to GANs and ViTs are presented in Section E.

Algorithm 1: Multilayer perceptron staining

Input : Trained multilayer perceptron network $\mathcal{N}$
Index k of neuron to stain in layer j
Detector response to trigger $\Delta \in \mathbb{R}$
Desired non-trigger detector response $\delta \in \mathbb{R}$
Whether the stain should be additive

Define the map ϕ from network inputs to inputs to layer j.
Let $W_j \in \mathbb{R}^{n \times m}$, $b_j \in \mathbb{R}^n$ be the layer j weight and bias.
Sample a detector weight vector $v \sim \mathcal{U}(\mathbb{S}^{m-1})$.
Optimise the trigger input $x^* \in \arg\max_{z \in S} v \cdot \phi(z)$.
if *stain is additive* **then**
  Let β be entry k of b_j and w be row k of W_j.
  Define $u = w + \frac{\Delta - \beta - w \cdot \phi(x^*)}{v \cdot \phi(x^*)} v$.
else
  Define $\beta = \delta$ and $u = \frac{\Delta - \beta}{v \cdot \phi(x^*)} v$.
Replace row k of W_j with u and entry k of b_j with β.

Output : Stained model $\mathcal{N}$ and trigger input x^*

Algorithm 2: Convolutional network staining

Input : Trained convolutional network $\mathcal{N}$
Parameters (k, j, Δ, δ) as in Alg. 1
Matrix to scalar reduction operation r
Whether layer j performs batch normalisation

Define the map $\phi : S = \mathbb{R}^{3 \times m_0 \times n_0} \to \mathbb{R}^{c_j \times m_j \times n_j}$ from network inputs to inputs to layer j.
Let $W_j \in \mathbb{R}^{c_{j+1} \times c_j \times \kappa_j \times \kappa_j}$ be the layer j weight, with kernel shape $\kappa_j \times \kappa_j$.
Sample a kernel $v \in \mathbb{R}^{c_j \times \kappa_j \times \kappa_j}$ from $\mathcal{U}(\mathbb{S}^{c_j \kappa_j^2 - 1})$, viewed as a tensor with shape $(c_j, \kappa_j, \kappa_j)$.
Optimise the trigger image $x^* \in \arg\max_{z \in S} r(v * \phi(z))$.
if *layer j performs batch normalisation* **then**
  Let $\mu_j, \sigma_j^2, w_j, b_j \in \mathbb{R}^{c_{j+1}}$ be the mean, variance, weight and bias of the layer j batch normalisation.
  Define $\alpha = \frac{(\Delta - \delta)(\sigma_j)_k}{(w_j)_k r(v * \phi(x^*))}$ and $\beta = \delta + \frac{(w_j)_k (\mu_j)_k}{(\sigma_j)_k}$.
else
  Let $b_j \in \mathbb{R}^{c_{j+1}}$ be the bias of layer j.
  Define $\alpha = \frac{(\Delta - \delta)}{r(v * \phi(x^*))}$ and $\beta = \delta$.
Replace $(W_j)_{k,\cdot,\cdot,\cdot}$ with αv, and entry k of b_j with β.

Output : Stained model $\mathcal{N}$ and trigger input x^*

3.1. Implementing standard staining schemas

The algorithms above introduce a novel training-free staining mechanism, and here we briefly discuss how it can be used to implement standard schemas from the literature.

Weight staining. Originally proposed by [7, 45], this schema encodes a secret message as a binary string, which is then implanted into the model as the signs of the weights of a target neuron (*e.g.* '-' encodes 0 and '+' encodes 1). The stain can be checked by decoding the message from the signs of the neuron's weight vector's components. This can be embedded into a non-additive stain by sampling the detector vector v from the orthant of the sphere with the specified

component signs, and the rest of the staining algorithm can then proceed as usual. This restriction on the sampling may increase the false positive rate of the detector (by decreasing the entropy of the distribution), but this should not present a problem if the dimension is sufficiently high.

Activation staining. This family of schemas, introduced in [28], instead embeds the secret message into the signs of the model's activations for a target input, rather than the weights. Our additive stain can be used to directly implement this. Suppose a detector vector is sampled for a given layer of the model, and a trigger input is optimised to produce a strong response to this detector vector. Then, this detector vector is weighted and added to each neuron in the layer, to ensure that the response to the trigger image from that neuron has the specified sign.

Output staining. Another standard schema family specifies (non-natural) class predictions for certain trigger input images [1, 8, 11, 32]. The stain can, therefore, be verified without access to model weights or activations. This can be implemented post-training by starting with a standard non-additive stain from Algorithm 1. If the stain detector neuron is implanted into row k of the weight matrix in layer j, then the detector's response (which will be zero for non-trigger inputs after a ReLU activation if $\delta \ll 0$) will multiply the entries of column k in layer $j+1$. For any chosen output class C, it is typically possible to find a vector (*e.g.* using gradient descent) which, if it were the output of layer $j+1$, would cause the model to confidently predict class C. Replacing column k of layer $j+1$ with this vector, therefore, causes the model to predict the chosen class for the trigger input; the selectivity of the detector ensures that the new column will not otherwise affect the model's performance. This technique extends analogously to convolutional stains.

4. Locking

Our locking methods build on the staining mechanism by inserting additional *disruptors* into the model to pollute its latent activations. The signal from the stain's detector neuron is used to disable the disruptors when the trigger is present in the input. Fig. 8 and Fig. 9 (in the supplementary material) show examples of images containing these patches. We show how the underlying structure of a convolutional detector can be exploited to produce a transferable trigger 'patch' which can be inserted into any image to unlock the full model performance. We present two variants of the locking mechanism. *Internal locking* embeds the lock into the weights of convolutional networks with a particular structure, while *squeeze-and-excite locking* uses a standard Squeeze-and-Excite block [21], common in computer vision models, as the foundation of a more generic procedure which can be added to any convolutional model.

Internal locking. Algorithm 3 adds an internal lock to a convolutional neural network. To present the idea in the simplest possible setting, we suppose that the model consists of a sequence of convolutional layers with biases (batch normalisation can be handled as in Algorithm 2), ReLU activations, followed by a single dense logits layer. For brevity, we do not explore here how the many additional model components present in practice may alter this algorithm: while they complicate the implementation, they do not change the fundamental concept. The algorithm consists of three main steps: implanting the detector and building the trigger (lines Line 1–4), propagating the detector signal through the model (lines 5–10), and building the disruptor (lines 11–14).

Algorithm 3: Convolutional network internal lock

Input : Parameters $(\mathcal{N}, k, j, \Delta > 0, \delta < 0)$ as in Alg. 2
Image coordinates (a, b) for the trigger patch
Disruption scale $s > 0$ and offset $t \in \mathbb{R}^c$

1 Define the map ϕ from network inputs to inputs to layer j.
2 Define the matrix to scalar map $r_{(a,b)}(M) := (M)_{a,b}$.
3 Stain $\mathcal{N}$ using Alg. 2, with the reduction map $r_{(a,b)}$.
4 Define the trigger patch π as the part of the trigger image x^* which is in the receptive field of $r_{(a,b)}(v * \phi(x^*))$.
5 To simplify presentation, swap kernels 1 and k in layer j.
6 **foreach** *convolution layer ℓ after layer j* **do**
7 Let $W_\ell \in \mathbb{R}^{c_{\ell+1} \times c_\ell \times \kappa_\ell \times \kappa_\ell}$, $b_\ell \in \mathbb{R}^{c_{\ell+1}}$ be the layer ℓ convolutional weight and bias.
8 Set $(W_\ell)_{1,\cdot,\cdot,\cdot} = 0 = (W_\ell)_{\cdot,1,\cdot,\cdot}$, and $(b_\ell)_1 = 0$.
9 Set $(W_\ell)_{1,1,\cdot,\cdot} = 1$.
10 **end**
11 Let $W_L \in \mathbb{R}^{c \times c_L}$, $b_L \in \mathbb{R}^c$ be the logits weight and bias.
12 Sample a disruptor vector u from $\mathcal{U}(\mathbb{S}^{c-1})$.
13 Let $\gamma \in \mathbb{R}^{c_L}$ be the logits layer input for model input x^*.
14 Set column 1 of W_L to $\frac{b_L - su + t}{(\gamma)_1}$; replace b_L with $su + t$.

Output : Locked model $\mathcal{N}$ and trigger patch π

The detector is built into a convolutional kernel as for the stain, but the trigger image is only optimised to produce a strong activation when the detector kernel is in a specific position (via the reduction map $r_{(a,b)}$). This has the effect that only a single small 'patch' of the input image (the receptive field of a single kernel in convolutional layer j) is optimised. Inserting this patch into (*e.g.* a corner of) an image during inference will therefore activate the detector.

Keeping this patch relatively small (to minimise its impact when inserted into an image) can be achieved by implanting the detector into a relatively early convolutional layer. Since the disruptor is disabled by the signal from the detector, the disruptor must be placed such that its receptive field always contains the trigger patch in the input image. For this reason, we place the disruptor into a late layer (the algorithm uses the model's logits layer, although many alternatives are possible), which depends on the entire input image, and build a conduit to propagate the detector signal to the disruptor (a more general technique to overcome this requirement is pre-

sented below). In Algorithm 3, this conduit is a sequence of 'identity' kernels, which simply propagate the signal in the detector channel (taken as channel 1 in the algorithm for simplicity) forwards through the model. Many techniques can be used to obfuscate the presence of this channel (and therefore the nature of the lock), although perhaps the simplest is just to add small random values to the modified weights.

The disruptor replaces the bias of the logits layer with a randomly sampled vector. This is disabled when the correct signal $(\gamma)_1$ is received from the detector through the conduit by restoring the original bias using the relevant column of the weight matrix. In practice, multiple interacting detectors and disruptors could be added to different places in the model, making it much more difficult to reverse-engineer the lock.

Squeeze-and-excite locking. Computer vision models for certain applications may not be suitable for internal locking if they lack good locations for a disruptor which can reliably respond to the detector signal. This is often the case in object detection, for example, where small objects may be detected using just a small portion of the image. The receptive field for these outputs may, therefore, simply not contain the trigger patch placed in the corner of the image, so its presence or absence would not influence the model's output. To overcome this challenge, we propose an alternative mechanism exploiting architectural blocks which normalise a convolution's response across the entire input image.

Squeeze-and-Excite (Sq-Ex) blocks [21] are a prototypical example of such an architecture, which is widely used in applications. The block is a map $s : \mathbb{R}^{c\times n\times m} \to \mathbb{R}^{c\times n\times m}$ which aggregates global spatial information about its input and uses this to rescale each channel. Specifically,

$$s(x) = x \odot q(x) \text{ with } q : \mathbb{R}^{c\times n\times m} \to \mathbb{R}^c, \qquad (1)$$
$$\text{where } q(x) = \sigma(S_2\rho(S_1\mu(x) + \tau_1) + \tau_2).$$

Here $\odot$ represents channel-wise multiplication, $\mu : \mathbb{R}^{c\times n\times m} \to \mathbb{R}^c$ computes the channel-wise mean, $S_1 \in \mathbb{R}^{d\times c}$ (typically with $d < c$) and $S_2 \in \mathbb{R}^{c\times d}$ are learned weight matrices, $\tau_1 \in \mathbb{R}^d$ and $\tau_2 \in \mathbb{R}^c$ are learned biases, and ρ and σ are the ReLU and sigmoid activation functions.[1]

Such blocks provide the key ability to propagate the detector signal *laterally* through the model, thereby enabling much more flexibility in the placement of disruptors. For example, if the detector produces value of 0 when the trigger patch is not present, and $\Delta > 0$ when it is, the averaged response will similarly be non-zero only when the trigger patch is present. A disruptor can, therefore, be embedded within the Sq-Ex block itself (as shown in Algorithm 4), or the block can be used to propagate the unlocking signal to disruptors later in the model. The algorithm only assumes that the convolutional layer j has a bias term, uses ReLU activation, and is followed by an Sq-Ex block. This lock is independent of the architecture of the rest of the model. To present the idea in a simple setting, the algorithm embeds the disruptor into the parameters S_2 and τ_2 of the Sq-Ex block itself, and S_1 is simply used as a conduit for the signal. In some cases, it can be helpful to stabilise the detector unlocking signal γ by combining the output of two or more ReLU neurons in the S_1 layer, *e.g.* to produce a hard sigmoid [20].

[1] Piecewise-polynomial variants of these are sometimes used, *e.g.* in MobileNetv3 [20] and the associated form of SSDLite [29, 38].

Algorithm 4: Squeeze-and-excite lock

Input : Parameters $(\mathcal{N}, k, j, \Delta, \delta, a, b, s, t)$ as in Alg. 3
Define the map ϕ from network inputs to inputs to layer j.
Stain $\mathcal{N}$ using Alg. 2, with the reduction map $r_{(a,b)}$.
Define the trigger patch π as the part of the trigger image x^* which is in the receptive field of $r_{(a,b)}(v * \phi(x^*))$.
To simplify presentation, swap kernels 1 and k in layer j.
Let S_1, τ_1, S_2, τ_2 be the Sq-Ex block parameters (1).
Set $(S_1)_{\cdot,1} = 0 = (S_1)_{1,\cdot}$, $(S_1)_{1,1} = 1$ and $(\tau_1)_1 = 0$.
Sample a disruptor vector u from $\mathcal{U}(\mathbb{S}^{c_{j+1}-1})$.
Let $\gamma = \mu(W_j * \phi(x^*) + b_j)$, *i.e.* the output of μ in (1)
Set column 1 of S_2 to $\frac{\tau_2 - su + t}{(\gamma)_1}$; replace τ_2 with $su + t$.
Set $(W_{j+1})_{\cdot,1,\cdot,\cdot} = 0$ (W_{j+1} is the layer $j+1$ weight).
Output : Locked model $\mathcal{N}$ and trigger patch π

Since the Sq-Ex block is used to propagate the detector signal spatially, the lock can be made independent of the architecture of the rest of the model. Special-purpose Sq-Ex blocks can be added to existing models to implement the lock, even if they were not already present. This can be achieved without significant impact on the performance of the model by sampling S_1, S_2, τ_1 and τ_2 randomly (as demonstrated experimentally in Sec. 6). By ensuring that the entries in S_1 and S_2 are relatively small, the overall scaling factor produced by the (unlocked) block can be made close to constant, and can therefore be counteracted by re-scaling the weights in the next layer. The computational cost of evaluating the additional Sq-Ex block is also significantly lower than that of the convolutional layers, since spatial information is aggregated to just a single value per channel. Moreover, if the Sq-Ex block is used to deactivate disruptors placed elsewhere in the model, simply pruning a suspicious Sq-Ex block would not deactivate the lock.

Extensions. Extensions to locking algorithms for GANs and ViTs are presented in Section E.

5. Theoretical guarantees

We provide two bounds on the worst-case false-positive rate of the randomly sampled detector neuron. Theorem 1 is geometric in nature, and exploits the fact that feature vector clouds typically have just a few dominant principal directions in practice. By projecting this cloud onto the random detector direction, we can bound the probability of sampling a detector and an input which leads to a false positive — *i.e.* a

detector response above some fixed threshold Δ. In essence, this bound shows which geometric aspects of feature space are important for building models which can be successfully stained and locked: a high feature space dimension, and feature vectors with a low PCA dimension. In such a setting, it is very unlikely that a detector vector and piece of input data will be sampled independently, which respond strongly to each other. The proof of this result is given in Appendix C.1.

Theorem 1 (Geometric bound on false positive rate) *Let S and T be sets, and consider a map $\mathcal{N} : S \to T$ such that $\mathcal{N} = \psi \circ \phi$ for two maps $\phi : S \to \mathbb{R}^d$ and $\psi : \mathbb{R}^d \to T$. Suppose that a detector vector $w \sim \mathcal{U}(\mathbb{S}^{d-1})$ is sampled uniformly from the sphere. Suppose that test data x are independently sampled from a distribution $\mathcal{D}$ on S such that $\mathbb{E}_x[\phi(x)] = \mu$ and $\mathrm{Cov}(\phi(x))$ has eigenvalues $\{\lambda_i\}_{i=1}^d$. Then, for any $\Delta > \|\mu\|$, the probability of sampling a detector and data point such that the detectors's response to the data point is stronger than Δ is bounded by*

$$P(w \sim \mathcal{U}(\mathbb{S}^{d-1}), x \sim \mathcal{D} : w \cdot \phi(x) > \Delta) \leq \frac{\sum_{i=1}^d \lambda_i}{2(\Delta - \|\mu\|)^2} \frac{d-1}{d+1} \left(\frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d+1}{2})} \right)^2. \tag{2}$$

Theorem 2 complements this with a data-driven bound on the false positive rate of a fixed detector directly in terms of observations from data. The bound can always be exactly evaluated from data observations to provide a guaranteed upper bound on the detector's worst-case false positive rate. This bound does not depend on any (typically unknown) properties of the data distribution, and evaluating it using more data will produce a tighter bound. An interesting feature of the result is that the smallest upper bound that it can produce depends only on the number of data samples available; when no false positives are observed in the sample, $n = 0$, and so the upper bound depends only on m, the number of samples used. Unlike Theorem 1, the bound of Theorem 2 does not consider the sampling of the detector weight vector. Together, these properties make Theorem 2 an ideal tool for certifying the false positive rate *of a fixed detector* which has perhaps already been implanted into a model. This result is proved in Appendix C.2.

Theorem 2 (Data-driven bound on false positive rate) *Let S and T be sets, and consider a map $\mathcal{N} : S \to T$ such that $\mathcal{N} = \psi \circ \phi$ for two maps $\phi : S \to \mathbb{R}^d$ and $\psi : \mathbb{R}^d \to T$. Let $w \in \mathbb{S}^{d-1}$ and $\Delta \in \mathbb{R}$ be fixed. Suppose that m test data points $x_1, \ldots, x_m$ are independently sampled from a distribution $\mathcal{D}$ on S. Of these m data points, suppose that exactly $n \leq m$ of them satisfy $w \cdot \phi(x_i) > \Delta$. Then, the probability of sampling a data point to which the detector's response is stronger than Δ is bounded by*

$$P(x \sim \mathcal{D} : w \cdot \phi(x) > \Delta) \leq 1 - \sup_{\epsilon > 0} \left(\frac{m-n}{m} - \epsilon \right) (1 - 2e^{-2m\epsilon^2}). \tag{3}$$

6. Experimental results

We demonstrate the practical performance of our staining and locking algorithms through a range of experimental results. We focus on a variety of widely used models for image classification and object detection, specifically ResNet50 [18] and VGG16 [39] for image classification, and SSDLite [29] with a MobileNet-v3 backbone [20] and Faster-RCNN [36] with a ResNet50 backbone [24] for object detection. These experiments were computed using pre-trained models available through PyTorch [3]. ImageNet [37] is used to assess the performance of image classification models, and COCO [27] is used for object detection models. We report each model's performance as the accuracy for image classification, and COCO average precision (AP) and average recall (AR) at IoU 0.5:0.95 for object detection, evaluated using the whole validation set of each benchmark. The original model's performance is also reported as a baseline. The results of each experiment are reported as the average over 50 samples of the detector per layer for image classification, and 40 for object detection (due to the increased computational cost of evaluating the model), and standard deviations are reported as shaded regions on the performance plots. The original model's performance is also reported as a baseline. To give a fair assessment, we avoided tuning the parameters used for each example, which in our experience can significantly improve the performance of the stained/unlocked models. Fig. 10–Fig. 12 (in the supplementary material) show specific examples of the performance of a locked object detection model. These experiments used the CREATE HPC facility at King's College London [22].

Staining. We inserted non-additive stains individually into several convolutional layers of each model using Algorithm 2. In each case, the neuron with the least ℓ^1 norm weight vector was replaced with the detector neuron. Fig. 1 (ResNet50) and Fig. 2 (SSDLite) present the results. Similar results for VGG16 (Fig. 13) and Faster-RCNN (Fig. 14) are included in the supplementary materials. No false positive detections (*i.e.* non-trigger images with a stronger response to the detector neuron than the trigger image) occurred in any of the experiments. We plot the distribution of responses of validation data to the randomly sampled detector (v in Algorithms 1 and 2) for each layer in each model, along with the rates guaranteed by Theorems 1 and 2. The range of response strengths for the optimised trigger input x^* obtained across all detector samples is given in the figure legends. The theoretical bounds on the response strength distributions were evaluated using 2,000 images sampled from the validation

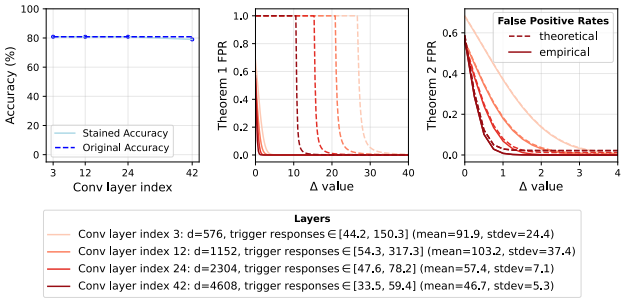

Figure 1. Staining ResNet50

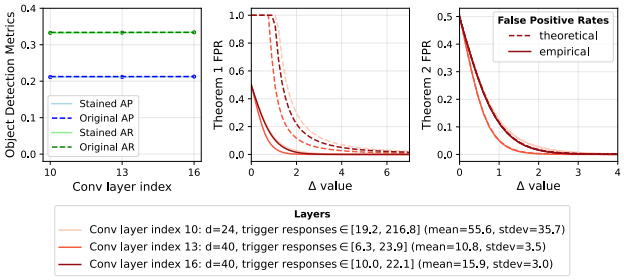

Figure 2. Staining SSDLite

set, using non-overlapping convolutional detector positions for each image (50 per image for Theorem 1, all for Theorem 2); the empirical response distributions were computed using all non-overlapping positions for the remaining images in each validation set. We observe the excellent agreement between theory and practice, particularly for Theorem 2. Recall, however, that the bounds from the two results are not directly comparable because they use different definitions of the false positive rate, depending on whether the detector neuron is considered to be sampled or fixed.

Internal locking. We added internal locks to the ResNet50 and VGG16 models, with the detector placed in a range of convolutional layers and a single disruptor placed in the logits layer. The results for ResNet50 are shown in Fig. 3. Similar results for VGG-16 (Fig. 15) can be found in the supplementary materials. The optimised trigger patch was taken to be the receptive field of the convolutional detector when in position (0,0). Although this overlaps with the convolution's padding region, the performance remains excellent. We report the models' performance in 6 settings: the original model, an 'edited' model in which the lock is implanted except for the disruptor itself, and the locked model; in each case, we evaluated using images with and without the trigger patch. It is clear from these results that a lock can be inserted into a pre-trained model using our algorithms with minimal impact on performance. There is a balance to be struck when placing the lock: too early in the model and the dimension of the feature space can be too low to produce

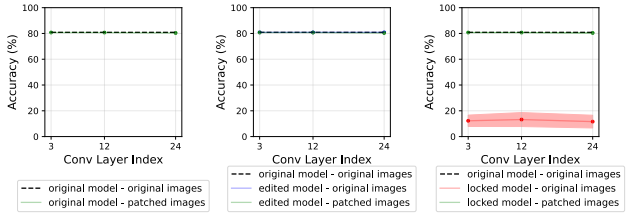

Figure 3. Internal lock for ResNet50

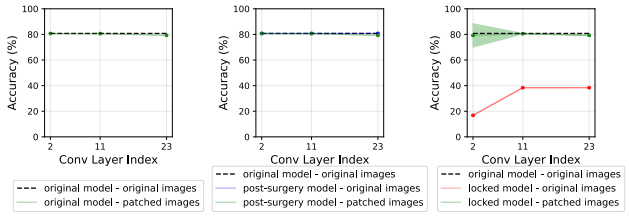

Figure 4. Squeeze-and-excite lock for ResNet50

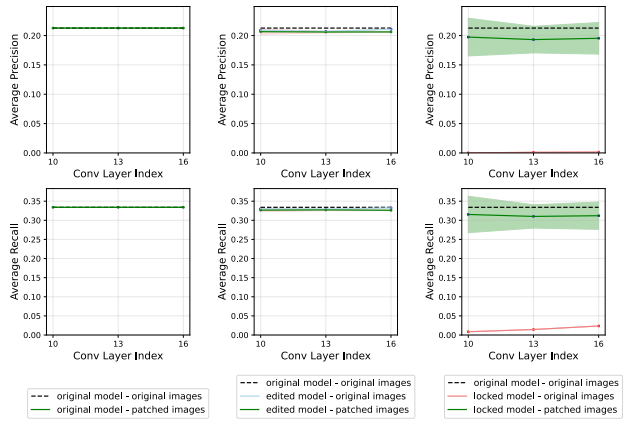

Figure 5. Squeeze-and-excite lock for SSDLite

a reliable detector; too late in the model, and the large size of the detector's receptive field (and hence the trigger patch added to images) can damage the model's performance.

Squeeze-and-excite locking. We demonstrate squeeze-and-excite locking in two forms: using the existing Sq-Ex blocks in the SSDLite model, and by adding new Sq-Ex blocks to each of the other models. Fig. 4 (ResNet50), Fig. 16 (VGG16), Fig. 5 (SSDLite) and Fig. 17 (Faster-RCNN) report the performance of each model in each of the 6 settings described in the previous section. The detector is placed in a range of convolutional layers, and the Sq-Ex block immediately follows it. The results clearly demonstrate both that this is an effective form of lock, and that a new block can be added to a pre-trained model without impacting the model's performance. The performance of the unlocked models is close to that of the original model in all cases. Where performance differences are observed (such as for

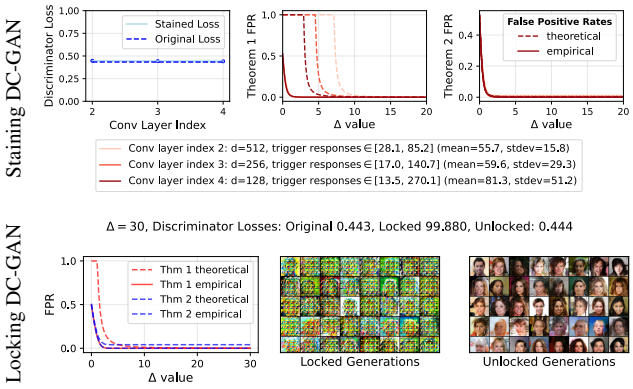

Figure 6. Staining and Locking DC-GAN

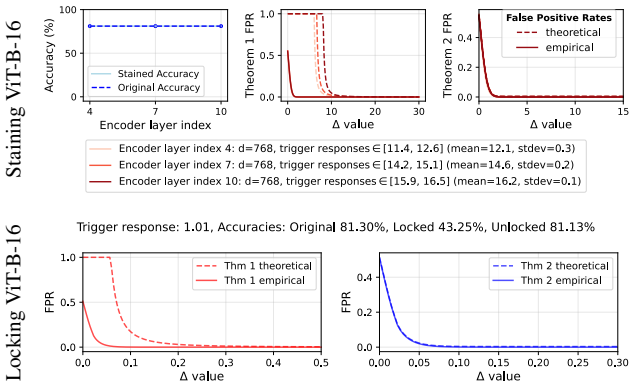

Figure 7. Staining and Locking ViT-B-16

the Faster-RCNN model in Fig. 17), this appears to be due to challenges in disabling the disruptor rather than because of the addition of the Sq-Ex block to a pre-trained model (shown by comparing the performance of the 'edited' model – containing the extra block but not the disruptor – against that of the 'unlocked' model).

GAN staining and locking. Figure 6 shows experimental results for staining and locking a DC-GAN [34] model for generating human faces. This was performed using the algorithmic extensions presented in Appendix E.1, and further examples of image generations are shown in Figure 18. Full details of the experimental setup are provided in Appendix E.1.1.

ViT staining and locking. Figure 7 shows experimental results for staining and locking a ViT [12] model for image classification. These were obtained using the algorithms and experimental setup described in Appendix E.2.

7. Discussion

Staining and locking are computationally cheap. The staining and locking mechanisms we present here are very cheap to implement. The model's weights are directly modified, so no training or fine-tuning is required. The algorithms also require no training or validation data, meaning that models trained on sensitive data can be easily handled. The inference-time cost of the stain and lock are minimal too, being embedded within the model itself, and even the Sq-Ex block is much cheaper than a single convolutional layer.

Forging attacks. In a forging attack [5], an attacker plants their own stain into a model and uses it to claim ownership. Since previous staining algorithms have required stains to be added at training time, this has not been considered a significant threat in the past. Our work reveals that functionally identical stains can easily be implanted post-training (Sec. 3.1), and forging attacks are therefore easy to perform.

Pruning attacks. An attacker may attempt to remove a stain or lock by simply pruning it out [5, 23]. This is not a risk for our additive stains, since the detector is simply added to the original neuron, so the normal behaviour of the original neuron is retained. A non-additive stain may result in a non-responsive neuron which could be pruned, depending on the values selected for δ and Δ. For the lock, on the other hand, pruning out the detector neuron would not help an attacker: without the ability to produce the correct unlocking signal, the full model performance could not be recovered. The disruptor is also difficult to remove, since it replaces the original bias vector which cannot be recovered without knowing the value of γ, the unlocking signal. The security properties of our stains and locks, including their robustness to standard attacks, are discussed further in Sec. F.

Obfuscating locks. We believe it is advantageous to encourage a variety of hiding and obfuscation techniques to grow around the fundamental building blocks we have provided here for locking. However, in our experience the task of detecting (and removing) a lock from model weights – which are widely known to be largely uninterpretable to the human eye – is already a significant challenge after only mild attempts at hiding it. This provides an asymmetrically difficult task for a thief with little work from the owner.

8. Conclusion

We have introduced new staining and locking algorithms for protecting pre-trained neural network models for computer vision applications. Unlike approaches in the literature, our stains and locks can be implemented without re-training or fine-tuning the model, and do not even require access to training or validation data. We have shown how several standard staining schemas can be reimplemented using our techniques. Novel and computable theoretical guarantees on the false positive rate of our stains and locks provide users with new tools for assessing the suitability of their models for staining and locking. Experiments using standard pre-trained models for image classification and object detection demonstrate the practical performance of our algorithms.

Acknowledgement

I.T. was supported by a grant, provided by the Ministry of Economic Development of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000C313925P4G0002) and the agreement with the Ivannikov Institute for System Programming of the Russian Academy of Sciences dated June 20, 2025 No. 139-15-2025-011.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: watermarking deep neural networks by backdooring. In *Proceedings of the 27th USENIX Conference on Security Symposium*, page 1615–1631, USA, 2018. USENIX Association. 1, 2, 4

[2] Manaar Alam, Sayandeep Saha, Debdeep Mukhopadhyay, and Sandip Kundu. NN-Lock: A lightweight authorization to prevent IP threats of deep learning models. *J. Emerg. Technol. Comput. Syst.*, 18(3), 2022. 2

[3] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 2024. 6

[4] Keith Ball. An elementary introduction to modern convex geometry. In *Flavours of Geometry*, 1997. 9

[5] Franziska Boenisch. A systematic review on model watermarking for neural networks. *Frontiers in Big Data*, 4, 2021. 1, 2, 8

[6] Yuling Cai, Fan Xiang, Guozhu Meng, Yinzhi Cao, and Kai Chen. AuthNet: Neural network with integrated authentication logic, 2024. 1, 2

[7] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. DeepMarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, page 105–113, New York, NY, USA, 2019. Association for Computing Machinery. 1, 2, 3

[8] Huili Chen, Bita Darvish Rouhani, and Farinaz Koushanfar. BlackMarks: Blackbox multibit watermarking for deep neural networks, 2019. 1, 2, 4

[9] Xiaojie Chu, Liangyu Chen, and Wenqing Yu. Nafssr: Stereo image super-resolution using nafnet. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1239–1248, 2022. 5

[10] Qi Cui, Ruohan Meng, Chaohui Xu, and Chip-Hong Chang. Steganographic passport: An owner and user verifiable credential for deep model ip protection without retraining. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12302–12311, 2024. 2

[11] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. DeepSigns: An end-to-end watermarking framework for protecting the ownership of deep neural networks. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2019. 1, 2, 4

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 8, 5

[13] A. Dvoretzky, J. Kiefer, and J. Wolfowitz. Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. *The Annals of Mathematical Statistics*, 27(3):642 – 669, 1956. 3

[14] Lixin Fan, Kam Woh Ng, Chee Seng Chan, and Qiang Yang. DeepIPR: Deep neural network ownership verification with passports. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6122–6139, 2022. 1, 2

[15] Guanhao Gan, Yiming Li, Dongxian Wu, and Shu-Tao Xia. Towards robust model watermark via reducing parametric vulnerability. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4728–4738, 2023. 2

[16] Yifeng Gao, Yuhua Sun, Xingjun Ma, Zuxuan Wu, and Yu-Gang Jiang. Modellock: Locking your model with a spell. In *Proceedings of the 32nd ACM International Conference on Multimedia*, page 11156–11165, New York, NY, USA, 2024. Association for Computing Machinery. 1, 2

[17] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2018. 1, 2

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 6

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 8

[20] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for MobileNetV3. In *ICCV*, pages 1314–1324. IEEE, 2019. 5, 6

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. 2, 4, 5

[22] King's College London. King's Computational Research, Engineering and Technology Environment (CREATE), 2024. 6

[23] Yue Li, Hongxia Wang, and Mauro Barni. A survey of deep neural network watermarking techniques. *Neurocomputing*, 461:171–193, 2021. 1, 2, 8

[24] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, and Ross Girshick. Benchmarking detection transfer learning with vision transformers, 2021. 6

[25] Yuxuan Li, Sarthak Kumar Maharana, and Yunhui Guo. Not just change the labels, learn the features: Watermarking deep neural networks with multi-view data. In *Computer Vision – ECCV 2024*, pages 273–289, Cham, 2025. Springer Nature Switzerland. 2

[26] Zheng Li, Chengyu Hu, Yang Zhang, and Shanqing Guo. How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN. In *Proceedings of the 35th Annual Computer Security Applications Conference*, page 126–137, New York, NY, USA, 2019. Association for Computing Machinery. 1, 2

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. 6

[28] Hanwen Liu, Zhenyu Weng, and Yuesheng Zhu. Watermarking deep neural networks with greedy residuals. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6978–6988. PMLR, 2021. 1, 2, 4

[29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. 5, 6

[30] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 8

[31] P. Massart. The tight constant in the dvoretzky-kiefer-wolfowitz inequality. *The Annals of Probability*, 18(3):1269–1283, 1990. 3

[32] E. Le Merrer, P. Pérex, and G. Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32:9233–9244, 2019. 1, 2, 4

[33] Xin Mu, Yu Wang, Zhengan Huang, Junzuo Lai, Yehong Zhang, Hui Wang, and Yue Yu. Encryip: A practical encryption-based framework for model intellectual property protection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(19):21438–21445, 2024. 2

[34] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 8, 4

[35] Yehonathan Refael, Adam Hakim, Lev Greenberg, Tal Aviv, Satya Lokam, Ben Fishman, and Shachar Seidman. SLIP: Securing LLMs IP using weights decomposition, 2024. 2

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. 6

[37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. 6

[38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, Los Alamitos, CA, USA, 2018. IEEE Computer Society. 5

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 6

[40] Z. Sun, R. Sun, C. Liu, A. Chowdhury, L. Lu, and S. Jha. ShadowNet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1596–1612, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 2

[41] Oliver Sutton, Qinghua Zhou, Wei Wang, Desmond Higham, Alexander N. Gorban, Alexander Bastounis, and Ivan Y Tyukin. Stealth edits to large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 2

[42] Ruixiang Tang, Mengnan Du, and Xia Hu. Deep serial number: Computational watermark for dnn intellectual property protection. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science and Demo Track*, pages 157–173, Cham, 2023. Springer Nature Switzerland. 2

[43] Ivan Y. Tyukin, Desmond J. Higham, and Alexander N. Gorban. On adversarial examples and stealth attacks in artificial intelligence systems. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2020. 2

[44] Ivan Y Tyukin, Desmond J Higham, Alexander Bastounis, Eliyas Woldegeorgis, and Alexander N Gorban. The feasibility and inevitability of stealth attacks. *IMA Journal of Applied Mathematics*, 89(1):44–84, 2023. 2

[45] 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*, page 269–277, New York, NY, USA, 2017. Association for Computing Machinery. 1, 2, 3

[46] Jiangfeng Wang, Hanzhou Wu, Xinpeng Zhang, and Yuwei Yao. Watermarking in deep neural networks via error back-propagation. *Electronic Imaging*, 32(4):22–1–22–1, 2020. 1, 2