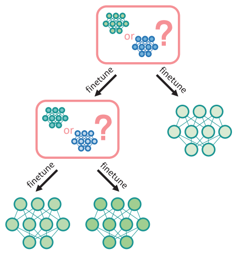
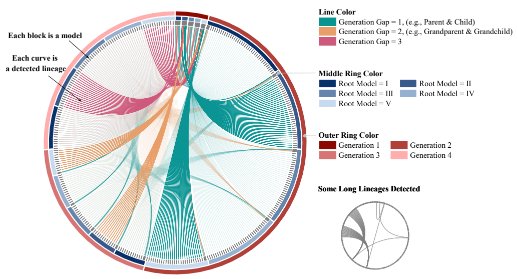
Neural Lineage

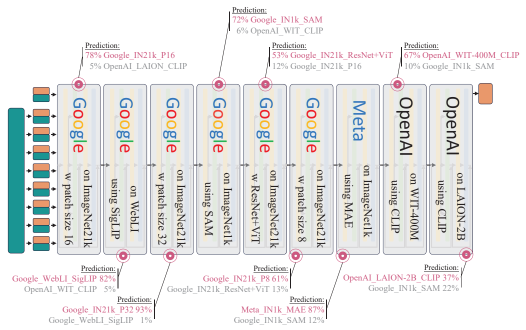
Runpeng Yu   Xinchao Wang[†]
National University of Singapore
r.yu@u.nus.edu   xinchao@nus.edu.sg



(a) Task illustration



(b) Model-level lineage detection



(c) Layer-level lineage detection

Figure 1. Neural lineage detection showcase. Fig. 1a shows the idea of neural lineage detection. Fig. 1b presents the detected lineages among 300 networks using the proposed lineage detector. Fig. 1c presents the lineage detector's ability to find ancestors of sub-networks.

Abstract

Given a well-behaved neural network, is possible to identify its parent, based on which it was tuned? In this paper, we introduce a novel task known as neural lineage detection, aiming at discovering lineage relationships between parent and child models. Specifically, from a set of parent models, neural lineage detection predicts which parent model a child model has been fine-tuned from. We propose two approaches to address this task. (1) For practical convenience, we introduce a learning-free approach, which integrates an approximation of the finetuning process into the neural network representation similarity metrics, leading to a similarity-based lineage detection scheme. (2) For the pursuit of accuracy, we introduce a learning-based lineage detector comprising encoders and a transformer detector. Through experimentation, we have validated that our proposed learning-free and learning-based methods outperform the baseline in various learning settings and are adaptable to a variety of visual models. Moreover, they also exhibit the ability to trace cross-generational lineage, identifying not only parent models but also their ancestors.

1. Introduction

Over the past decade, deep learning technology has undergone rapid evolution, amassing vast research and applications. Today, the development of deep learning exhibits several new phenomena. First, the "Pre-Training-Finetuning" framework has supplanted "Training from Scratch".Deep learning models are no longer isolated; downstream-task models have established a significant dependency relationship with pre-trained models. Secondly, with the construction of "Machine Learning as a Service" (MLaaS), extensive deep learning communities, and vast neural network model repositories have been established. Numerous deep-learning models tailored for various applications have been developed. These changes indicate that we are now creating a network of deep learning models characterized by a vast number, differentiation, diversity, and strong inter-generational dependencies among models. We are now, poised to consider how to study or leverage this burgeoning network of deep learning models. Foremost among the tasks is to distinguish the generational relationships between models and reveal this underlying network describing the connections between models.

We define the task of model lineage detection, illustrated in Fig. 1a, and investigate it in this work. Specifically, given a model f_c, performing model lineage detection for f_c involves identifying a model f_p, from which f_c is fine-tuned[1], among a set of candidate models $\{f_p^{(m)}\}_{m=1}^M$. We collectively refer to the candidate models $f_p^{(m)}$'s as parent models, and the model f_c for which we need to perform lineage detection as a child model.

[†] Corresponding author.

[1] In this work, we always consider tuning all the parameters.

This task holds significant practical importance in the realms of model reuse, intellectual property protection, and model regulation. For the users of a model, neural lineage detection aids in understanding the model's knowledge inherited from parent models, revealing its generalizability and robustness, uncovering its potential fairness and bias issues, and, thus, enabling more targeted adaptation and improvement of the model. For model developers, the generational relationships provided by neural lineage detection can serve as evidence for intellectual property protection. For third-party organizations, neural lineage detection enhances model traceability, becoming a tool for accountability and regulation.

We propose one learning-free and one learning-based method to detect neural lineage. The first, the learning-free method, discussed in Sec. 3, identifies the most probable parent-child model pair through model similarity. While directly measuring the distance between parent and child models using model similarity metrics is feasible, this approach does not leverage one key piece of information: in neural lineage detection, the child model is fine-tuned from the parent model. Instead, based on the Neural Tangent Kernel (NTK) theory, we utilize neural network linearization to approximate the fine-tuning process and identify a proximate child model. We then compare the distance between this approximated child model and the actual one. This idea can be applied across various commonly used similarity metrics to establish a set of model similarity metrics embedded with an approximation of the fine-tuning process. Additionally, we address the computational complexity of neural network linearization using the Taylor expansion method, which is summarized in Prop. 1.

The second, training-based approach, discussed in Sec. 4, trains a neural network classifier that incorporates multiple encoder blocks to encode the parameters and (or) features of both parent and child networks, and a prediction block based on the transformer architecture to output the probability of the child model being fine-tuned from each parent neural network. Unlike the learning-free method, which explicitly estimates the fine-tuning process, this learning-based approach relies on the neural network's ability to learn the changes that the fine-tuning process imparts on the parameters and features.

The contributions of this work are:

- We explore a novel and significant task, namely neural lineage detection, aiming at detecting the parent model from which a child model is fine-tuned.
- We propose two methodologies: a learning-free approach, which can be used for any model and at any time, and a learning-based one with higher prediction accuracy.
- Experimental results demonstrate the versatility of our method across classification, segmentation, and detection models; in scenarios of few-shot learning and data imbalance; and for cross-generational lineage detection.

Illustrative Examples. Fig. 1b visualizes the lineage detection results for 300 models using the proposed Lineage Detector. Notably, not only all parent-child pairs were accurately identified (plotted with green curves), but all the cross-generational grandparent-child pairs and great-grandparent-child pairs are also accurately identified (plotted with orange and red curves, respectively).

In training downstream task models, it's common to assemble multiple pre-trained models and then fine-tune, to amalgamate the functionalities of different models. We constructed a specific case of such "assembling+finetuning" strategy to test whether our proposed lineage detector can be applied to sub-network lineage detection. We concatenate transformer layers from different ViT-B models in the timm repository to create a hybrid ViT-B model. This model was then fine-tuned on the CIFAR-10 dataset. Subsequently, the lineage detector was tasked with identifying the origin of each transformer layer. Fig. 1c displays the results, where the origins of all transformer layers were accurately identified.

2. Related Works

2.1. Deep Model IP Protection

The task most similar to neural lineage detection is deep model IP protection, which enables model owners to identify pirated copies of their protected models. Common methods for deep model IP protection include: (1) Model watermarking, where a detectable IP identifier is embedded into the neural network through finetuning or retraining [1, 6, 8, 13, 32, 38, 41, 43, 44, 49, 58, 67, 71, 74]. (2) Model fingerprinting, which involves using gradient optimization or retrieval to find a set of conferrable samples that induce similar behaviors in the source and pirated models, while ensuring distinct behaviors between the source model and irrelevant models [4, 40, 45, 52, 55, 62, 68, 73].

The primary distinctions between neural lineage detection and neural network IP protection are:

1. IP protection involves a binary classification to determine whether a model is pirated, while neural lineage detection is a multi-class task to identify parent-child pairs.
2. Solutions for neural network IP protection often depend on external media, like IP identifiers from model watermarking or conferrable samples through model fingerprinting, which entail extra training, optimization, or search processes. In contrast, neural lineage detection seeks to directly identify model lineage without extra encoding, modification, or fingerprint of the model.
3. IP protection research also considers that model pirates might modify the stolen models, expecting the protection to resist modifications, such as fine-tuning. Con-

ceptually, IP protection methods claim the ability to detect even fine-tuned pirated models. However, the definition of finetuning in the context of IP protection differs significantly from that in neural lineage detection. In neural lineage detection, finetuning is defined as its general practice: given a model for dataset $\mathcal{D}_1$, fine-tuning adapts it to perform the task on dataset $\mathcal{D}_2$, with the requirement that the fine-tuned model performs well on $\mathcal{D}_2$. In contrast, IP protection views fine-tuning as an attack strategy, aiming at circumventing IP protection without compromising the performance on $\mathcal{D}_1$. This type of fine-tuning attack does not prioritize the performance on $\mathcal{D}_2$, often employing minimal learning rates, few epochs, or a subset of $\mathcal{D}_1$ as $\mathcal{D}_2$, which differs fundamentally from the general practice of fine-tuning [1, 4, 13, 16, 27, 36, 37, 53].

2.2. Neural Network Representation Similarity

Measuring representation similarity is a fundamental task of deep learning, and numerous methods have been developed. Methods based on Canonical Correlation Analysis (CCA) address the high dimensionality of neural network features [48, 56, 69, 70], while alignment-based methods resolve inconsistencies in feature dimensions [20, 39, 65]. Beyond directly measuring the distance between feature matrices (vectors) or their statistics [59, 63], instance-wise feature similarity can be leveraged to construct similarity matrices [30, 75], graphs [22], and simplicial complexes [3, 28, 57], or to identify neighbors for each instance in the feature space. Subsequently, the distance between these constructed objects can be measured, or the neighbor information can be utilized for similarity aggregation. The similarity of neural networks is also linked to model fine-tuning. Many studies can be viewed as explicitly constraining certain similarity metrics to enhance stability during model fine-tuning or continual learning, thereby reducing the distance between child and parent models [11, 18, 21, 42]. In our learning-free method, we do not directly employ previous similarity metrics. Instead, we integrate an estimation of the fine-tuning process into previous similarity metrics and use this new series of metrics to achieve neural lineage detection.

2.3. Neural Network Linearization

The study of neural network linearization originated with the Neural Tangent Kernel (NTK) [12, 26] and has been theoretically proven that, as its width increases, the network can be effectively approximated by its first-order Taylor expansion with respect to its parameters at initialization [34]. Numerous empirical studies have also demonstrated that the linear approximation of neural networks can match or even exceed the performance of nonlinear neural networks, particularly in small-sample regions or specially designed problems [2, 51]. Neural network linearization has become a viable alternative to training neural networks in fields such as pruning at initialization [17, 19, 46, 60], training-free neural architecture search [7, 47, 61], training time prediction [72], dataset distillation [50], and deep model ensemble [23], directly approximating the results of gradient optimization. Our learning-free method can also be seen as an application of this idea. However, beyond this, our proposed method further addresses the efficiency issues associated with the use of neural network linearization.

3. Similarity-Based Detection

3.1. Notation

Let $\mathcal{D}$ denote the training dataset of the finetuning process and $\mathcal{X} = \{\boldsymbol{x} : (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}\}$ and $\mathcal{Y} = \{\boldsymbol{y} : (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}\}$ denote the inputs and the targets, respectively. Let N denote the number of samples in $\mathcal{D}$ and $\boldsymbol{x}^{(i)}$ and $\boldsymbol{y}^{(i)}$ denote the i-th input and target, respectively. Let $f_p : \mathcal{X} \to \mathcal{Y}$ and $f_c : \mathcal{X} \to \mathcal{Y}$ denote the parent neural network and the child neural network parameterized by $\boldsymbol{\theta}_p$ and $\boldsymbol{\theta}_c$, respectively. We use the vectorization form of both the parameters of the neural network and the output of the neural network to facilitate the matrix representation. For example, we write $\boldsymbol{\theta}_p \in \mathbb{R}^{|\boldsymbol{\theta}_p| \times 1}$, where $|\boldsymbol{\theta}_p|$ is the number of the parameters in f_p, $f_p(\boldsymbol{x}) \in \mathbb{R}^{K \times 1}$, where K is the dimension of the targets. Given that we are considering the matching problem between the child model and multiple parent models, we use superscripts to index the parent models. Specifically, $f_p^{(m)}$ is used to denote the m-th parent model. Let $\bar{f}_p : \mathcal{X} \to \mathcal{Y}$ denote the linear approximation of the fine-tuned model, which will be defined in Sec. 3.2. Let $\langle \cdot, \cdot \rangle$ denote the matrix inner product. Let $\mathrm{sg}(\cdot)$ denote the stop gradient operator. Let $*$ denote the element-wise multiplication.

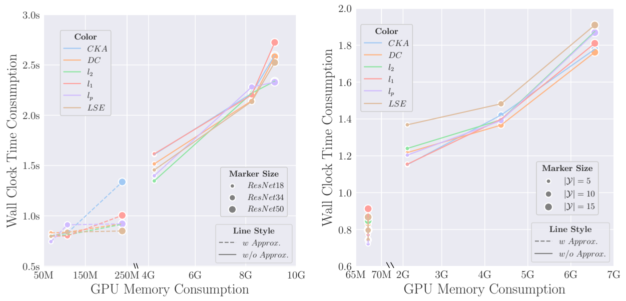
3.2. Method

The learning-free method, conceptually, consists of two steps: (1) approximation, which approximates the child model obtained through the fine-tuning process, and (2) measurement, which compares the distance between the approximated child model and the actual child model.

In the approximation step, our method uses the strategy of neural network linearization. Previous work has demonstrated that for wide neural networks, this linear approximation provides an excellent and efficient approximation of the dynamics of the original non-linear neural network under gradient descent algorithms [35]. Specifically, given a parent model and a child model, the output of the linearized parent model is defined by its first-order Taylor expansion

$$\bar{f}_p(\boldsymbol{x}) \triangleq f_p(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p), \tag{1}$$

where $f_p(\boldsymbol{x})$ denotes the output of the parent model;



(a) Different model size (b) Different output size

Figure 2. Comparison of the execution time and GPU memory Consumption for methods with and without approximation.

$\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x})$ yields the Jacobian matrix of $f_p(\boldsymbol{x})$ with respect to its parameter $\boldsymbol{\theta}_p$; and $\boldsymbol{\theta}_c - \boldsymbol{\theta}_p$ signifies the parameter change between the child and parent models. If the child model is indeed fine-tuned from the parent model, then the approximated $\bar{f}_p(\boldsymbol{x})$ should closely resemble the output of the child model itself, $f_c(x)$. If the child model is not fine-tuned from the parent model, the gradient and parameter change in equation Eq. (1) would be entirely independent.

In the measurement step, we compute the similarity between $\bar{f}_p(\boldsymbol{x})$ and $f_c(\boldsymbol{x})$. However, to obtain $\bar{f}_p(\boldsymbol{x})$, one must first compute the Jacobian matrix. Moreover, most neural network similarity metrics require multiple samples. Thus, for calculating a scalar distance metric, first computing $\bar{f}_p(\boldsymbol{x})$ and then measuring the distance necessitates gradient computations through the entire neural network $N \times K$ times, and the storage of gradient value amounts to $N \times K \times |\boldsymbol{\theta}|$, where N is the number of samples and $|\boldsymbol{\theta}|$ is the number of parameters in f_p. To reduce the computational time and space complexity, instead of this step-by-step approach, we consider integrating the first approximation step with the second distance measurement step with a linear approximation of the similarity metric. We apply this idea to several commonly used similarity metrics and list the approximation results below. Detailed derivations are left in the appendix.

Proposition 1 (Similarity Approximation.) *Define* $\boldsymbol{d}_i \triangleq f_p(\boldsymbol{x}^{(i)}) - f_c(\boldsymbol{x}^{(i)})$ *to be the difference between the outputs of* f_p *and* f_c *for input* $\boldsymbol{x}^{(i)}$. *Let* $s(\bar{f}_p, f_c)$ *be the similarity between* $\{\bar{f}_p(\boldsymbol{x}^{(i)})\}_{i=1}^N$ *and* $\{f_c(\boldsymbol{x}^{(i)})\}_{i=1}^N$, *whose linear approximation has the form of*

$$s(\bar{f}_p,f_c) \approx s(f_p,f_c) + \nabla_{\boldsymbol{\theta}_p}\Big[\sum_{i=1}^{N} sg(\Pi_i) f_p(\boldsymbol{x}^{(i)})\Big](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) \quad (2)$$

Part 1 *(l_1* ***Similarity.****)* *For l_1 similarity* $s(\bar{f}_p, f_c) \triangleq -\frac{1}{NK} \sum_{i=1}^N ||\bar{f}_p(\boldsymbol{x}^{(i)}) - f_c(\boldsymbol{x}^{(i)})||_1$, $\Pi_i = -\frac{1}{NK} sign(d_i)^T$*;*

Part 2 *(l_2* ***Similarity.****)* *For l_2 similarity* $s(\bar{f}_p, f_c) \triangleq -\frac{1}{NK} \sum_{i=1}^N ||\bar{f}_p(\boldsymbol{x}^{(i)}) - f_c(\boldsymbol{x}^{(i)})||_2^2$, $\Pi_i = -\frac{2}{NK} \boldsymbol{d}_i^T$*;*

Part 3 *(l_p* ***Similarity.****)* *For l_p similarity* $s(\bar{f}_p, f_c) \triangleq -\frac{1}{NK} \sum_{i=1}^N ||\bar{f}_p(\boldsymbol{x}^{(i)}) - f_c(\boldsymbol{x}^{(i)})||_p^p$,

$$\Pi_i = -\frac{p}{NK}[sign(\boldsymbol{d}_i) * |\boldsymbol{d}_i|^{p-1}]^T; \quad (3)$$

Part 4 *($\log$-sum-exp* ***Similarity.****)* *For LSE similarity* $s(\bar{f}_p, f_c) \triangleq -\frac{1}{NKt} \sum_{i=1}^N \log \sum_{k=1}^K e^{t|\bar{f}_p(\boldsymbol{x}^{(i)}) - f_c(\boldsymbol{x}^{(i)})|}$,

$$\Pi_i = -\frac{1}{NK}[sign(\boldsymbol{d}_i) * softmax(t|\boldsymbol{d}_i|)]^T; \quad (4)$$

Part 5 (CKA.) *For squared Centered Kernel Alignment with linear kernel defined in [30],* $\Pi_i = s(f_p, f_c)\zeta_i$*;*

Part 6 (DC.) *For squared Distance Correlation defined in [75],* $\Pi_i = s(f_p, f_c)\xi_i$*;*

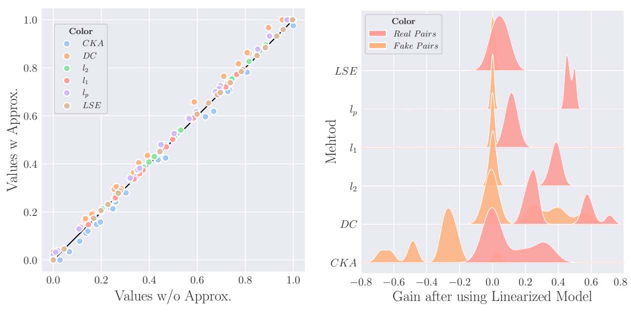
where the complete expression of ζ_i and ξ_i are presented in the Appendix.

Remark 1 *The approximations in Prop. 1 reduce the computational complexity. Computing the approximated similarity metrics necessitates only a single back-propagation through the neural network, and each neural network parameter requires the storage of just one gradient value. Fig. 2 compares the wall-clock time and GPU memory consumption for computing model similarity using our method versus a step-by-step approach. Our method significantly reduces GPU memory consumption and also requires less computation time. Notably, since our approach first computes a weighted sum of model outputs before evaluating the derivative, GPU memory consumption does not increase with the size of the model's output.*

Remark 2 *In Prop. 1, the gradient calculations in the previous step-by-step method have been consolidated. All approximated similarity metrics solely encompass one gradient computation of one scalar. This scalar is a weighted sum of the neural network's output, where different similarity metrics employ distinct weighting schemes. For instance, the π_i of l_2 similarity is the difference in output between the parent and child models, suggesting that l_2 similarity focuses more on features with significant discrepancies between the parent and child models, while down-weighting features that are similar between them. A similar observation can be made for the approximation of the $\log$-sum-exp similarity, where Π_i is the softmax score of the difference in output between the parent and child models.*

Remark 3 *Given that l_∞ is inherently non-differentiable, it precludes its Taylor expansion. The p-norm and $\log$-sum-exp in Prop. 1 are approximations to the l_∞ similarity.*

Remark 4 *We also empirically assessed the accuracy of the approximation in the proposition, with relevant results*



(a) Similarity values comparison (b) Similarity gain density

Figure 3. The influence of approximations and linearized model. Fig. 3a plots the similarity values measured with and without using approximation. Fig. 3b plots the similarity gains after using the linearized model to get a better anchor.

displayed in Fig. 3a. The x-axis of Fig. 2(a) represents the similarity values computed using the step-by-step approach, while the y-axis represents the similarity values obtained using the method in Prop. 1. All data points are closely located around the identity line, indicating that the error introduced by the Taylor expansion in the proposition is minimal. Therefore, considering the reduction in computational complexity brought about by the approximation, it offers a more advantageous approach overall compared to the step-by-step method.

Thus far, given a child model f_c for lineage detection and a set of parent models $\{f_p^{(i)}\}_{i=1}^M$, we have the similarity metrics $\{s(\bar{f}_p, f_c)\}_{m=1}^M$ based on the linear approximation of the fine-tuning. From it, a matching probability vector $P \in [0,1]^M$ can be calculated as the softmax of the similarity measures with $P_m \triangleq \frac{e^{s(\bar{f}_p^{(m)}, f_c)}}{\sum_{l=1}^M e^{s(\bar{f}_p^{(l)}, f_c)}}$. Finally, the prediction can be drawn by selecting the parent model with the highest matching probability.

To enhance the adaptability across diverse models, we introduce two adjustments in our approach. First, recognizing that not every model aligns seamlessly with the theoretical underpinnings of NTK, we substitute the fixed Eq. (1) with $\bar{f}p(\boldsymbol{x}) \triangleq f_p(\boldsymbol{x}) + \alpha \nabla \boldsymbol{\theta}_p f_p(\boldsymbol{x})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)$, where α is a hyper-parameter, analogous to the learning rate. Second, we extend the approximation and similarity measurement to the intermediate layer features, rather than confining them solely to the neural network's final output.

3.3. Intuitive Explanation

Proposition 2 (The Optimality of Measurement) *Let $f_p'(\boldsymbol{x}) \triangleq f_p(\boldsymbol{x}) + \boldsymbol{W}(\boldsymbol{x})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)$ and $f_p''(\boldsymbol{x}) \triangleq f_p(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x})\boldsymbol{Z}$ denote two linearized approximation of parent model with undetermined parameters $\boldsymbol{W}(\boldsymbol{x}) \in \mathbb{R}^{K \times |\boldsymbol{\theta}|}$ for each $\boldsymbol{x}$ and $\boldsymbol{Z} \in \mathbb{R}^{|\boldsymbol{\theta}| \times 1}$ for all $\boldsymbol{x}$, respectively. For the real parent-child model pair (f_p, f_c) and l_2 similarity,*

$$s(\bar{f}_p, f_c) = \max_{\boldsymbol{W}(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}} s(f_p', f_c) = \max_{\boldsymbol{Z}} s(f_p'', f_c) \quad (5)$$

This proposition demonstrates that, when allowing for a linear approximation of the finetuning process, under the l_2 similarity metric, our method yields the optimal results, maximizing the similarity between the real parent-child pair. To elaborate further, the first optimization problem suggests that for each $\boldsymbol{x}$, our method implicitly identifies a hyperplane passing through $f_p(\boldsymbol{x})$ such that the projection of $f_c(\boldsymbol{x})$ onto this hyperplane minimizes the distance to the point $(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)$ on the hyperplane; the second optimization problem indicates that our method first implicitly determines the projection of $\{f_c(\boldsymbol{x}^{(i)})\}_{i=1}^N$ onto the hyperplane constructed by $\{\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})\}_{i=1}^N$, and then measures the distances between $\{f_c(\boldsymbol{x}^{(i)})\}_{i=1}^N$ and its projection.

Given that the aforementioned proposition only theoretically analyzed how linearization in Eq. (1) impacts the similarity between real parent-child pairs, we further empirically verify if the intuition derived from the proposition holds true, and provide insights into how linearization affects the similarity between fake parent-child pairs. Fig. 3b illustrates the changes in similarity after linearization. For real parent-child pairs, the similarity metric generally increases, which is consistent with our theoretical findings. For fake parent-child pairs, the linearized model causes two kinds of phenomenon: it either generally reduces the similarity or has minimal impact on the similarity metric. The reasons for these phenomena might be that the linearized model essentially predicts the fine-tuning process starting from the parent model based on the gradient direction and uses the predicted new model as an anchor to compare its similarity with the child model. If the angle between the gradient generated by the fake parent model and the actual fine-tuning direction is obtuse, the prediction of fine-tuning results in an anchor even farther from the child model than the previous fake parent, leading to a decrease in similarity. If the gradient generated by the fake parent model is orthogonal to the actual fine-tuning direction, the distance from the new anchor to the child model remains almost the same as the distance from the previous parent model to the child model, resulting in an almost unchanged similarity metric.

4. Lineage Detector

In this section, we introduce the learning-based lineage detection method. The proposed lineage detector can be viewed as a nonlinear improvement of the method described in Sec. 3. Here, the effects of finetuning are directly learned by the lineage detector, rather than being approximated by a fixed linear relationship. The lineage detector, shown in Fig. 4, comprises a weight encoder, a feature encoder, and a transformer detector, which takes the weights and features

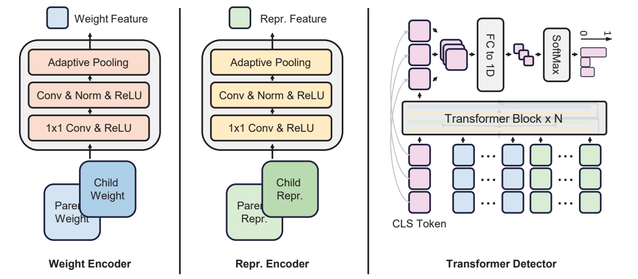


Figure 4. The architecture of the proposed lineage detector. Weights and features are first encoded separately and are then fed into a transformer to obtain the prediction score.

of parent and child models as input to predict their lineage probability. The workflow of the lineage detector is detailed as follows.

First, parameters $\boldsymbol{\theta}_p$ and features $\boldsymbol{F}_p$ from a parent model are first vectorized and then reshaped into matrices, such that, $\boldsymbol{\theta}_p \in \mathcal{R}^{H_\theta \times W_\theta}$ and $\boldsymbol{F}_p \in \mathcal{R}^{H_F \times W_F}$. The widths and heights are chosen based on the architecture of the parent model. The weights and features of a child model are also pre-processed to have the same shapes, so that, $\boldsymbol{\theta}_c \in \mathcal{R}^{H_\theta \times W_\theta}$ and $\boldsymbol{F}_c \in \mathcal{R}^{H_F \times W_F}$. The weights from the parent and child are then stacked together as input for the weight encoder, while their features are similarly stacked as the input of the feature encoder. We define the stacked weights and features as $\boldsymbol{\theta} \triangleq [\boldsymbol{\theta}_p; \boldsymbol{\theta}_c] \in \mathcal{R}^{2 \times H_\theta \times W_\theta}$ and $\boldsymbol{F} \triangleq [\boldsymbol{F}_p; \boldsymbol{F}_c] \in \mathcal{R}^{2 \times H_F \times W_F}$, respectively. Compared to encoding the parent's weights(features) and the child's weight(feature) independently, this stacking strategy facilitates the model's direct learning of the element-level relationships between parent and child models, since the parent's and child's weights(features) are naturally aligned and the finetuning process can bring subtle changes in the weights(features) that can be missed if encoded separately.

The weight encoder $\Phi_{\boldsymbol{\theta}}$ and the feature encoder $\Phi_{\boldsymbol{F}}$ share the same CNN architecture. In the first layer, 1x1 kernels are used to further facilitate the model's extraction of the element-level relationships. The intermediate layers of the convolutional network are standard convolutions combined with batch normalization and ReLU activation. After the convolutional layers, adaptive average pooling, with output size 1×1, is used to address the potential inconsistencies in the shape of the $\boldsymbol{\theta}$ and $\boldsymbol{F}$. Squeezing the redundant dimension, the outputs of the encoders are vectors, $z_{\boldsymbol{\theta}} \triangleq \Phi_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \in \mathcal{R}^D$ and $z_{\boldsymbol{F}} \triangleq \Phi_{\boldsymbol{F}}(\boldsymbol{F}) \in \mathcal{R}^D$, where D is the number of the output channels of the CNN and the token size of the following transformer Ψ.

Similar to the position embedding used in transformers, we add learnable weight embedding $E_{\boldsymbol{\theta}} \in \mathcal{R}^D$ and feature embedding $E_{\boldsymbol{\theta}} \in \mathcal{R}^D$ to distinguish weight and feature for subsequent processing by the transformer. These are then concatenated with a learnable classification (cls) token $z_{cls} \in \mathcal{R}^D$ and fed into a multi-layer transformer. A transformer is used because the weights and features change interdependently during the finetuning process, and the attention mechanism allows the final prediction to consider both the information from weights and features and their interactive effects. Furthermore, the transformer structure is adaptable to scenarios with multiple weight tokens, multiple feature tokens, or only one of the weight and feature tokens. Finally, a linear head transforms the final output at the position of the class token into the final score, which can be written as

$$z^{final} = \Psi([z_{cls}; z_\theta + E_{\boldsymbol{\theta}}; z_F + E_{\boldsymbol{F}}]), \quad (6)$$

$$s = Head(z_{cls}^{final}). \quad (7)$$

For a set of parent models $\{f_p^{(m)}\}_{m=1}^M$, a set of score $\{s^{(m)}\}_{m=1}^M$ can be obtained for each parent-child pair, and the model lineage probability prediction can be derived from the softmax of these scores. Given the true parent index, the lineage detector can be trained using a cross-entropy loss.

5. Experiments

In this section, we present our core experimental results, with more results and implementation details reported in the appendix.

5.1. Classification

Basic setup. The basic setup focuses on classification. We examined two architectures: a 3-layer fully connected(FC) network with ReLU activation and no normalization, and ResNet18 [24]. The simple FC network was included in the experiments as it better aligns with the theoretical hypotheses of Neural Network Linearization, allowing us to observe the performance of the learning-free method under relatively ideal conditions. For the FC network, 20 networks trained from scratch on MNIST [33] or 12 on CIFAR100 [31] were used as parent models. For ResNet, 7 networks pre-trained on ImageNet [14] in timm [64] package were used. Child models were generated by finetuning parent models on downstream datasets with varying hyperparameters and random seeds. The downstream datasets included FMNIST [66], EMNIST-Letters [10], CIFAR10 [31], Pet [54], and DTD [9]. The average number of child models is 204 per dataset-network structure pair. We compared the accuracy of lineage detection of the proposed learning-free and learning-based methods against a baseline that directly uses parent-child model similarity. For training the lineage detector, child models were randomly split into training, validation, and test sets in a 7:1:2 ratio, and the performance on the test set was recorded and averaged over 5 repetitions. The results for both learning-free and learning-based methods are averages from 5-fold experiments.

			FMNIST	EMNIST	Cifar10	SVHN	DTD	Pet	Cifar10 5shot	Cifar10 50shot	Cifar10 Imbalanced
Fully Connected Network	w/o Approx.	l_1	24.11±1.27	35.97±1.25	70.04±1.05	84.09±1.64	-	-	80.31±1.22	73.93±1.01	75.42±1.43
		l_2	23.98±2.04	45.61±1.59	76.88±0.64	68.84±1.71	-	-	86.37±1.02	75.32±1.18	78.24±1.72
		l_∞	11.84±1.23	37.99±1.83	77.32±0.81	70.06±2.25	-	-	78.79±1.44	75.14±1.08	76.41±1.24
		CKA	4.39±0.37	6.58±0.76	52.49±1.31	29.92±1.27	-	-	18.18±1.83	21.21±0.97	50.76±1.72
		DC	3.51±0.49	6.32±0.91	54.92±1.22	29.92±0.87	-	-	19.71±1.71	21.81±1.52	50.38±1.61
	w Approx.	l_1	25.44±1.27	36.40±1.31	71.59±1.12	85.22±1.57	-	-	85.61±2.51	74.62±1.12	75.76±1.51
		l_2	26.75±1.75	46.93±1.73	80.03±0.71	70.07±1.71	-	-	88.64±1.31	75.38±1.18	78.79±1.72
		l_p	21.05±1.69	41.23±1.79	78.79±0.83	70.83±1.61	-	-	86.73±1.41	69.71±1.51	76.51±1.16
		LSE	26.32±1.27	37.28±1.36	73.48±0.78	71.59±1.79	-	-	85.61±1.39	75.76±1.05	78.03±1.47
		CKA	5.71±0.48	14.91±1.04	53.03±1.16	57.95±0.98	-	-	21.21±2.41	21.97±0.96	51.14±1.69
		DC	14.48±1.41	7.64±0.81	61.74±0.91	28.41±1.01	-	-	25.01±2.42	36.74±1.89	51.14±1.41
	Lineage Detector		**97.86±1.19***	**93.61±2.37***	**99.11±0.36***	**99.31±0.35***	-	-	**99.94±0.35***	**98.84±1.43***	**99.35±0.24***
ResNet18	w/o Approx.	l_1	90.37±0.82	90.47±0.84	90.19±1.33	98.75±0.72	89.41±1.66	90.23±0.64	98.85±0.19	98.68±0.88	87.50±1.11
		l_2	86.84±1.15	87.51±1.36	90.10±1.41	98.64±0.22	88.68±1.66	89.43±0.99	97.75±0.19	98.41±0.41	87.68±0.11
		l_∞	78.57±0.87	95.83±0.65	79.76±0.64	95.24±0.41	88.24±2.01	85.81±1.45	98.07±0.69	98.93±0.51	79.17±1.21
		CKA	68.45±2.49	75.59±2.19	45.83±0.81	63.54±1.93	62.94±1.52	66.04±1.98	71.11±3.45	64.82±0.52	62.51±1.48
		DC	58.92±2.21	32.14±1.61	61.33±2.71	45.43±1.68	62.94±1.75	51.47±3.17	67.78±2.28	65.56±0.51	60.47±1.59
	w Approx.	l_1	98.81±0.41	98.92±0.59	94.64±0.63	98.81±0.41	94.71±0.32	95.27±0.84	99.89±0.62	98.93±0.61	95.84±0.99
		l_2	88.69±0.65	88.09±1.08	90.47±0.84	97.94±1.26	90.59±1.75	90.53±0.63	98.89±0.59	99.36±0.52	88.09±0.93
		l_p	80.95±0.48	80.95±1.16	81.55±0.68	94.05±0.92	91.18±1.87	89.66±0.35	92.22±1.86	93.54±2.23	80.95±0.85
		LSE	96.43±0.97	98.21±0.41	89.29±1.21	99.41±0.33	13.53±1.43	11.24±1.21	98.82±0.22	97.85±1.21	91.67±1.23
		CKA	70.23±2.54	77.38±1.93	47.62±0.94	64.88±1.89	66.47±0.83	68.05±1.61	72.22±3.25	68.81±3.71	66.08±2.42
		DC	60.11±2.33	34.52±0.85	63.09±2.73	45.83±1.47	63.12±1.43	52.07±2.73	67.89±2.18	65.58±2.64	60.71±1.09
	Lineage Detector		**99.21±0.16**	**99.32±0.05**	**99.87±0.07***	**99.64±0.12**	**99.59±0.47***	**99.74±0.21***	**99.91±0.52**	**99.38±0.02**	**97.01±0.41**

Table 1. Lineage detection results for classification tasks. The best (the second-best) ones are marked in **bold** (underlined). * indicates that the lineage detector has a statistically significant improvement compared to the best among other methods, with a p-value less than 0.05.

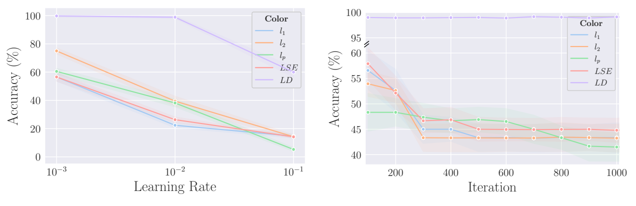
The results are shown in Tab. 1. Key observations include: (1) Generally, the learning-based method significantly outperforms the others with an average accuracy improvement of 36.26%, and the learning-free method generally outperforms the baseline by an average of 3.17%, with a maximum improvement of 28%. (2) The performance of the learning-free method depends on the similarity metric used. No single metric consistently outperforms others, but generally, norm distances perform better than advanced neural network similarity metrics, possibly because these neural network similarity metrics filter out differences that can be eliminated through projection or rotation [30, 75], which are often key clues for lineage detection.

Few-shot and imbalanced data. We also tested our methods on CIFAR10 -5-shot, -50-shot, and -imbalanced datasets[2]. The scale of these experiments was comparable to the previous CIFAR10 experiments. Overall, both learning-free and learning-based methods performed better in few-shot scenarios than on complete datasets, likely due to the quicker convergence and fewer samples, making the finetuning process easier to approximate or learn. The advantages of the proposed methods persisted on the few-shot and imbalanced datasets.

Learning rate and iteration. Further analysis was conducted on the effects of different finetuning learning rates and iterations on our methods. This experiment was motivated by the direct impact of learning rate and finetuning iterations on the distance between parent and child models, thus influencing the difficulty of lineage detection. In Fig. 5, we plotted the accuracy of lineage detection for ResNet18 models on CIFAR10 as a function of learning rate and iteration. As the learning rate or iteration increases, lineage detection becomes more challenging, with all methods showing a decreasing tendency in accuracy. However, the learning-based method proves more robust compared to the learning-free ones.

Across generation. Similar to the number of iterations, another factor influencing the difficulty of lineage detection is the number of generations of finetuning. We defined the 7 ResNet18 parent models as the Generation 1 (G1) models and fine-tuned them down 3 generations along EMNIST-Letters(G2)→FMNIST(G3)→EMNIST-Balanced[10](G4). Each generation produced a comparable number of child models to previous experiments. We conduct lineage detection between all descendant-ancestor pairs, with results shown in Tab. 2. It can be observed that the relative advantages of the proposed methods exist under different generation gap sizes, with the learning-based method still performing the best, followed by the learning-free method.

[2]constructed by down-sampling the last 5 classes of CIFAR10 to 10%



(a) Different learning rate (b) Different iteration

Figure 5. Lineage detection result when finetuning ResNet18 on CIFAR10 with different learning rates and iterations.

		G2	G3	G4
	l_1 w/o Appx.	● 90.47±0.84	● 89.07±1.72	● 86.31±2.47
	CKA w/o Appx.	75.59±2.19	● 44.44±2.89	● 38.09±1.15
G1	l_1 w Appx.	98.92±0.59	● 97.02±0.51	● 96.83±0.82
	CKA w Appx.	77.38±1.93	● 51.59±2.98	● 51.19±1.27
	Lineage Detector	● 99.11±0.36	● 98.81±0.13	● 97.75±1.53
	l_1 w/o Appx.	-	● 91.27±1.89	● 83.93±2.67
	CKA w/o Appx.	-	● 65.88±2.63	● 67.86±2.03
G2	l_1 w Appx.	-	● 98.41±0.53	● 95.24±1.71
	CKA w Appx.	-	● 75.41±2.81	● 72.03±3.17
	Lineage Detector	-	99.61±0.83	● 98.38±0.02
	l_1 w/o Appx.	-	-	94.05±1.49
	CKA w/o Appx.	-	-	● 66.67±1.42
G3	l_1 w Appx.	-	-	● 97.02±0.75
	CKA w Appx.	-	-	● 72.03±1.91
	Lineage Detector	-	-	● 98.41±0.98

Table 2. Results of across-generational lineage detection.

However, as the generation gap increases, the general detection performance declines. The red dots in the table indicate the relative accuracy, with darker red representing lower accuracy. The dots in the diagonal blocks have the lightest red, corresponding to the smallest generation gap and highest accuracy, while the darkest dots are in the upper-right block, corresponding to the largest generation gap and lowest accuracy.

5.2. Other Tasks and Losses

Dense prediction. We also investigate the lineage detection performance in object detection and segmentation tasks. For detection, we used 8 models fine-tuned on a subset of the PASCAL [15] dataset as parent models from the officially released DETR [5] model, and 56 models further fine-tuned on a non-overlapping subset of PASCAL as child models. For segmentation, we used the officially released DETR panoptic segmentation model as the starting point, similarly fine-tuned on PASCAL across two generations to create parent and child models. Tab. 3 shows proposed methods maintain their advantages.

Regularizations. Finally, we examine the impact of introducing additional regularization losses during finetuning on lineage detection. We considered EWC regularization [29] on parameters and KL-divergence (KLD) regularization [25] on outputs, widely used in continual learning and knowledge distillation, respectively. These regularizations represent two settings: EWC directly constrains child model parameters from deviating too far from the parent model parameters, potentially strengthening the lineage relationship and can be considered as a strategy to facilitate lineage detection; KLD requires the fine-tuned model's output to closely resemble another teacher model's output (not the parent), potentially weakening the lineage relationship and can be considered as a strategy to attack lineage detection. The EWC experiment is conducted by adding EWC loss in the original FC+FMNIST setup. The KLD experiment is conducted by introducing an additional teacher model and KL divergence loss on top of the original ResNet18+FMNIST setup. Results in Tab. 3 show that the relative advantages between methods remained largely unchanged, with EWC leading to increased accuracy and KLD to decreased accuracy.

		Detection	Segmentation	Continual Learning	Knowledge Distillation
w/o Approx.	l_1	95.86±2.03	25.01±3.97	98.25±0.12	93.38±1.66
	l_2	91.22±2.26	22.91±2.11	99.74±0.38	92.92±1.59
	l_∞	78.58±4.01	33.33±4.68	97.81±0.54	70.65±1.21
	CKA	73.20±4.54	24.98±5.89	39.04±1.11	44.03±3.25
	DC	71.76±4.81	20.85±4.24	37.84±1.02	34.43±3.01
w Approx.	l_1	<u>96.42±1.23</u>	31.27±3.45	99.32±0.12	93.57±1.75
	l_2	92.85±1.91	25.01±1.52	**99.83±0.21**	<u>94.03±1.23</u>
	l_p	91.07±1.57	<u>35.41±2.29</u>	99.82±0.23	82.57±2.19
	LSE	96.23±1.83	22.93±2.31	99.11±0.38	93.21±1.71
	CKA	78.56±4.43	24.99±5.95	41.23±1.11	46.79±3.38
	DC	73.21±4.45	27.07±3.84	38.61±1.31	34.86±2.74
Lineage Detector		**99.28±0.72**	**37.56±4.04**	<u>99.18±0.63</u>	**99.07±0.07**

Table 3. Lineage detection results for other tasks and losses. The best (the second-best) ones are marked in **bold** (<u>underlined</u>).

6. Conclusion

In this paper, we propose the task of neural lineage detection, which aims to determine the parent-child relationship between models without relying on external media. To address this task, we first introduce a learning-free, similarity-based detection method that incorporates an approximation of the finetuning process into the model similarity measurement, achieving accuracy beyond directly using similarity measurement. We further propose a transformer-based lineage detector, which shows significant performance gains over the learning-free method.

Acknowledgment

This research is supported by the National Research Foundation, Singapore, under its AI Singapore Programme (AISG Award No: AISG2-RP-2021-023).

References

[1] Yossi Adi, Carsten Baum, Moustapha Cissé, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium*, 2018. 2, 3

[2] Sanjeev Arora, Simon S. Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2020. 3

[3] Serguei Barannikov, Ilya Trofimov, Nikita Balabin, and Evgeny Burnaev. Representation topology divergence: A method for comparing neural network representations. In *International Conference on Machine Learning*, 2022. 3

[4] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary. In *ACM Asia Conference on Computer and Communications Security*, 2021. 2, 3

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, 2020. 8

[6] 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 *International Conference on Multimedia Retrieval*, 2019. 2

[7] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective. In *International Conference on Learning Representations, ICLR 2021*. 3

[8] Xuxi Chen, Tianlong Chen, Zhenyu (Allen) Zhang, and Zhangyang Wang. You are caught stealing my winning lottery ticket! making a lottery ticket claim its ownership. In *Advances in Neural Information Processing Systems*, 2021. 2

[9] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014. 6

[10] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, 2017. 6, 7

[11] Andrew Cotter, Heinrich Jiang, Maya R. Gupta, Serena Lutong Wang, Taman Narayan, Seungil You, and Karthik Sridharan. Optimization with non-differentiable constraints with applications to fairness, recall, churn, and other goals. *Journal of Machine Learning Research*. 3

[12] Amit Daniely. SGD learns the conjugate kernel class of the network. In *NeurIPS*, 2017. 3

[13] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019. 2, 3

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, 2010. 8

[16] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In *Annual Conference on Neural Information Processing Systems*, 2019. 3

[17] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. 2023. 3

[18] Mahdi Milani Fard, Quentin Cormier, Kevin Robert Canini, and Maya R. Gupta. Launch and iterate: Reducing prediction churn. In *NeurIPS*, 2016. 3

[19] Thomas Gebhart, Udit Saxena, and Paul R. Schrater. A unified paths perspective for pruning at initialization. *ArXiv*, abs/2101.10552, 2021. 3

[20] Charles Godfrey, Davis Brown, Tegan Emerson, and Henry Kvinge. On the symmetries of deep learning models and their internal representations. In *NeurIPS*, 2022. 3

[21] Gabriel Goh, Andrew Cotter, Maya R. Gupta, and Michael P. Friedlander. Satisfying real-world goals with dataset constraints. In *NeurIPS*, 2016. 3

[22] M. Gwilliam and Abhinav Shrivastava. Beyond supervised vs. unsupervised: Representative benchmarking and analysis of image representation learning. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 3

[23] Bobby He, Balaji Lakshminarayanan, and Yee Whye Teh. Bayesian deep ensembles via the neural tangent kernel. In *NeurIPS*, 2020. 3

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016. 6

[25] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, 2015. 8

[26] Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018. 3

[27] Hengrui Jia, Christopher A. Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *USENIX Security Symposium*, 2021. 3

[28] Valentin Khrulkov and I. Oseledets. Geometry score: A method for comparing generative adversarial networks. In *International Conference on Machine Learning*, 2018. 3

[29] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017. 8

[30] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 3, 4, 7

[31] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[32] Yingjie Lao, Peng Yang, Weijie Zhao, and Ping Li. Identification for deep neural network: Simply adjusting few

weights! *IEEE International Conference on Data Engineering*, 2022. 2

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 6

[34] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *NeurIPS*, 2019. 3

[35] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *NeurIPS*, 2019. 3

[36] Suyoung Lee, Wonho Song, Suman Jana, Meeyoung Cha, and Sooel Son. Evaluating the robustness of trigger set-based watermarks embedded in deep neural networks. *IEEE Transactions on Dependable and Secure Computing*, 2023. 3

[37] Meng Li, Qi Zhong, Leo Yu Zhang, Yajuan Du, Jun Zhang, and Yong Xiang. Protecting the intellectual property of deep neural networks with watermarking: The frequency domain approach. In *International Conference on Trust, Security and Privacy in Computing and Communications*, 2020. 3

[38] Meng Li, Qi Zhong, Leo Yu Zhang, Yajuan Du, Jun Zhang, and Yong Xiang. Protecting the intellectual property of deep neural networks with watermarking: The frequency domain approach. In *International Conference on Trust, Security and Privacy in Computing and Communications*, 2020. 2

[39] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E. Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *ICLR*, 2016. 3

[40] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. Modeldiff: testing-based DNN similarity comparison for model reuse detection. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021. 2

[41] Yiming Li, Linghui Zhu, Xiaojun Jia, Yong Jiang, Shu-Tao Xia, and Xiaochun Cao. Defending against model stealing via verifying embedded external features. In *AAAI Conference on Artificial Intelligence*, 2022. 2

[42] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, 2016. 3

[43] Xiaoxuan Lou, Shangwei Guo, Jiwei Li, and Tianwei Zhang. Ownership verification of dnn architectures via hardware cache side channels. *IEEE Transactions on Circuits and Systems for Video Technology*, 2021. 2

[44] Sofiane Lounici, Mohamed Njeh, Orhan Ermis, Melek Önen, and Slim Trabelsi. Yes we can: Watermarking machine learning models beyond classification. *IEEE Computer Security Foundations Symposium*, 2021. 2

[45] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. In *International Conference on Learning Representations*, 2021. 2

[46] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023. 3

[47] Ji-Yoon Choi Ji-Hyeok Moon Young-Ilc Mok, Byunggook Na, Ji-Hoon Kim, Dongyoon Han, and Sungroh Yoon. Demystifying the neural tangent kernel from a practical perspective: Can it be trusted for neural architecture search without training? *CVPR*, 2022. 3

[48] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018. 3

[49] Yuki Nagai, Yusuke Uchida, Shigeyuki Sakazawa, and Shin'ichi Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 2018. 2

[50] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. In *NeurIPS*, 2021. 3

[51] Guillermo Ortiz-Jiménez, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. What can linearized neural networks actually say about generalization? In *NeurIPS*, 2021. 3

[52] Xudong Pan, Mi Zhang, Yifan Lu, and Min Yang. TAFA: A task-agnostic fingerprinting algorithm for neural networks. In *European Symposium on Research in Computer Security*, 2021. 2

[53] Xudong Pan, Yifan Yan, Mi Zhang, and Min Yang. Metav: A meta-verifier approach to task-agnostic model fingerprinting. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022. 3

[54] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. *CVPR*, 2012. 6

[55] Zirui Peng, Shaofeng Li, Guoxing Chen, Cheng Zhang, Haojin Zhu, and Minhui Xue. Fingerprinting deep neural networks globally via universal adversarial perturbations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 2

[56] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Narain Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NeurIPS*, 2017. 3

[57] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, I. Oseledets, and Emmanuel Müller. The shape of data: Intrinsic distance for data distributions. *ICLR*, 2019. 3

[58] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *ACM on International Conference on Multimedia Retrieval*, 2017. 2

[59] Chenxu Wang, Wei Rao, Wenna Guo, P. Wang, Jun Liu, and Xiaohong Guan. Towards understanding the instability of network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2020. 3

[60] Chaoqi Wang, Guodong Zhang, and Roger B. Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations, ICLR 2020*, 2020. 3

[61] Haoxiang Wang, Yite Wang, Ruoyu Sun, and Bo Li. Global convergence of MAML and theory-inspired neural architecture search for few-shot learning. In *CVPR*, 2022. 3

[62] Si Wang and Chip-Hong Chang. Fingerprinting deep neural networks - a deepfool approach. *IEEE International Symposium on Circuits and Systems*, 2021. 2

[63] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, 2020. 3

[64] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 6

[65] Alex H. Williams, Erin Kunz, Simon Kornblith, and Scott W. Linderman. Generalized shape metrics on neural representations. In *NeurIPS*, 2021. 3

[66] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 6

[67] Chenqi Xie, Ping Yi, Baowen Zhang, and Futai Zou. Deepmark: Embedding watermarks into deep neural network using pruning. *IEEE International Conference on Tools with Artificial Intelligence*, 2021. 2

[68] Kan Yang, Run Wang, and Lina Wang. Metafinger: Fingerprinting the deep neural networks with meta-training. In *International Joint Conference on Artificial Intelligence*, 2022. 2

[69] Xingyi Yang, Jingwen Ye, and Xinchao Wang. Factorizing knowledge in neural networks. In *European Conference on Computer Vision*, 2022. 3

[70] Xingyi Yang, Daquan Zhou, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. In *Advances in Neural Information Processing Systems*, 2022. 3

[71] Jingwen Ye, Songhua Liu, and Xinchao Wang. Partial network cloning. 2023. 2

[72] Luca Zancato, Alessandro Achille, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Predicting training time without training. In *NeurIPS*. 3

[73] Jingjing Zhao, Qin Hu, Gaoyang Liu, Xiaoqiang Ma, Fei Chen, and Mohammad Mehedi Hassan. Afa: Adversarial fingerprinting authentication for deep neural networks. *Computer Communications*, 2020. 2

[74] Xiangyu Zhao, Yinzhe Yao, Hanzhou Wu, and Xinpeng Zhang. Structural watermarking to deep neural networks via network channel pruning. *IEEE International Workshop on Information Forensics and Security*, 2021. 2

[75] Xingjian Zhen, Zihang Meng, Rudrasis Chakraborty, and Vikas Singh. On the versatile uses of partial distance correlation in deep learning. In *ECCV*, 2022. 3, 4, 7