Finding Lottery Tickets in Vision Models
via Data-driven Spectral Foresight Pruning

Leonardo Iurada   Marco Ciccone   Tatiana Tommasi
Politecnico di Torino, Italy
{leonardo.iurada, marco.ciccone, tatiana.tommasi}@polito.it

Abstract

Recent advances in neural network pruning have shown how it is possible to reduce the computational costs and memory demands of deep learning models before training. We focus on this framework and propose a new pruning at initialization algorithm that leverages the Neural Tangent Kernel (NTK) theory to align the training dynamics of the sparse network with that of the dense one. Specifically, we show how the usually neglected data-dependent component in the NTK's spectrum can be taken into account by providing an analytical upper bound to the NTK's trace obtained by decomposing neural networks into individual paths. This leads to our Path eXclusion (PX), a foresight pruning method designed to preserve the parameters that mostly influence the NTK's trace. PX is able to find lottery tickets (i.e. good paths) even at high sparsity levels and largely reduces the need for additional training. When applied to pre-trained models it extracts subnetworks directly usable for several downstream tasks, resulting in performance comparable to those of the dense counterpart but with substantial cost and computational savings. Code available at: *https://github.com/iurada/px-ntk-pruning*

1. Introduction

Almost daily we hear about new breakthroughs achieved by artificial intelligence. Most of them are obtained by powerful foundational models [3, 25, 38] that however require prohibitively high computational resources and energy costs. These issues raise critical concerns in terms of financial and environmental sustainability [32, 42] and pose significant challenges for future applications requiring lightweight and efficient models embedded in always-on devices and the Internet of Things (IoT).

Given the over-parameterized nature of modern deep neural networks, one solution to alleviate their resource demands involves removing a significant number of less im-
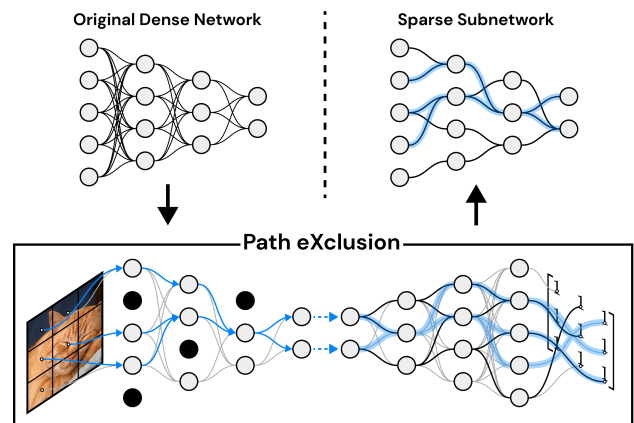

Figure 1. Our Path eXclusion (PX) involves two copies of the original dense network. One copy (bottom left) estimates data-relevant paths, depicted by blue arrows, and injects the extracted information into the other network (blue shading). The other copy (bottom right) evaluates path relevance in terms of parameter connections in the network, illustrated by black connections. These estimations are then combined to score each parameter, finding a subnetwork by retaining only the most relevant paths based on data, architecture, and initialization. The identified sparse subnetwork closely mimics the training dynamics of the original dense network.

portant neurons or connections. Several pruning approaches have been developed with the goal of lowering networks' complexity without sacrificing accuracy [17, 31, 46, 47, 50], and they can further benefit from efficient implementations of sparse primitives [11, 36] and hardware designed to exploit sparsity [45]. These methods are traditionally applied late in training or post-training with the goal of reducing inference time, but recent findings suggest that pruning can also be performed in advance [13].

Specifically, *Pruning at Initialization* (PaI) searches for randomly initialized subnetworks that once trained can match the test accuracy of the original dense networks with a largely reduced learning cost. Prior works have proposed PaI strategies based on the impact of each parameter on the loss [1, 30] or on different saliency metrics that estimate the information flow in the network [44]. Some recent publica-

tions have targeted the evaluation of the training dynamics based on the Neural Tangent Kernel Theory (NTK, [23]) to define the parameters' scores. Although showing promising results, they usually neglect [15] or loosely approximate [49] the data contribution to the NTK spectrum as they claim that data has a minimal impact on finding lottery tickets *i.e.* good paths in the network [13]. Some of these approaches also suffer for layer collapse *i.e.* the premature pruning of an entire layer that would make the network untrainable. As discussed in [44], its occurrence can be avoided under specific conditions.

One question that remains open is whether pruning can be applied to pre-trained networks before their downstream transfer [6, 22]. This is a crucial and timely problem as pre-trained models continue to grow in size, and pruning could be used to reduce the cost of fine-tuning on downstream tasks. This still defines a proper PaI setting where the initial model is trained on huge corpora and the goal is not only to compress it but also to preserve its transferability capabilities in the obtained subnetworks.

With our work we advance PaI research by proposing the following contributions:

- We present Path eXclusion (PX, see Figure 1.) a PaI method that estimates the relevance of each network's parameter to the training dynamic through a newly defined bound on the trace of the NTK.
  - The saliency function formulated from the bound guarantees that the network parameters have only positive scores. Together with the iterative nature of PX, this provides guarantees on avoiding layer collapse.
  - Differently from previous work, the relevance score of our PX depends jointly on the data, and on the native neurons and layer connections.
- We experimentally show that PX is not only robust to different architectures and tasks, but can be effectively used to search for subnetworks in large pre-trained models that maintain almost intact transferability.

2. Related Works

The question of how to significantly reduce the number of parameters of a neural network while maintaining its performance dates back to the 1980s [28]. Several strategies include matrix and tensor factorization [35], generalized dropout [41], and adding regularization terms in the learning objective [31, 50] to enforce sparse networks. Other approaches identify parameters with low magnitude after training and discard the corresponding connections [17, 18, 28, 46, 47]. For all these methods, the main goal is to improve test efficiency while the computational cost of training remains the same as that of a dense network.

In the last years, the focus has moved towards efficient training with one milestone provided by the *Lottery Ticket Hypothesis* [13]. It demonstrated that within overly large networks it is possible to identify *winning tickets*, *i.e.* smaller subnetworks that once trained perform nearly as well as their dense counterpart. The Iterative Magnitude Pruning (IMP) algorithm discovers these subnetworks through several rounds of alternated training and progressive pruning guided by the magnitude of the surviving parameters. Despite its effectiveness, the high computational costs of IMP led to the development of alternative cheaper methods for finding sparse networks. They are usually identified as *Pruning-at-Initialization* (PaI), or *foresight pruning* algorithms and can be organized into two main families.

The *data-agnostic* methods exploit either random or constant mini-batches to probe the network and score each parameter on the basis of its relevance to some network's property. Then, only a small fraction of the parameters with top scores is kept for training. SynFlow [44] builds on the hypothesis that the synaptic saliency for the incoming parameters to a hidden neuron is equal to the sum of the synaptic saliency for the outgoing ones. Thus it evaluates the importance of each parameter on the basis of its relation to those in the previous and following layers. SynFlow-L2 [15] scores each parameter by considering its contribution to the network's training dynamics estimated via the Neural Tangent Kernel theory [23]. LogSynFlow [4] rescales the scores of SynFlow to account for the possible issue of exploding gradients. NTK-SAP [49] improves the previous methods by exploiting a more precise estimate of the training dynamics defined from the full spectrum of the Neural Tangent Kernel and then discards parameters that contribute the least to it. All these approaches compute importance scores iteratively with multiple forward-backward passes over the network, while PHEW [37] introduces random walks biased towards higher parameter magnitudes and requires a single pruning round.

The *data-driven* methods assert the relevance of the data and of the learning task in evaluating the importance of each network's parameter when pruning and avoiding large degradation in model performance. SNIP [30] defines a saliency score for the parameters based on how they contribute to changing the initial loss. ProsPR [1] combines an estimate of the effect of pruning on the loss and on the meta-gradients that define the optimization trajectory. GraSP [48] takes the gradient norm after pruning as a reference criterion, and drops the parameters that result in its least decrease. These methods are single-shot, while some variants of SNIP such as IterSNIP [8] and FORCE [9] exploit iterative solutions to avoid layer collapse.

Our work falls in between the two families described. We build on the NTK theory already used by the data-agnostic approaches and we show how information from the data can be used to guide the pruning process with significant advantages in training efficiency. Indeed, as al-

ready discussed in [14, 21, 51] data independence can be considered as a limitation rather than a benefit. An intuitive reason is that data statistics have a crucial effect on some network components as batch normalization that contributes to the overall network behavior and parameter relevance [44]. Moreover, by focusing on the network training dynamics rather than on the loss, as most of the data-driven approaches do, our method proves to be task-independent, with the obtained sparse network remaining effective even when transferred to new downstream tasks.

3. Method

In this section, we start by describing the standard framework adopted by Pruning-at-Initialization (PaI) methods and the intuition of our foresight pruning algorithm designed to calculate and preserve the trace of the Neural Tangent Kernel (NTK). Afterward, we provide a brief overview of the theory underpinning NTK and present how to express its trace by exploiting the notion of network paths. Finally, we introduce our Path eXclusion (PX) algorithm that drops those network weights that minimally change the trace of the NTK, so that the obtained sparse network retains only the most relevant paths of the original dense network.

3.1. Problem Formulation

Let us consider a neural network $\boldsymbol{f} : \mathbb{R}^d \rightarrow \mathbb{R}^K$ parametrized by $\boldsymbol{\theta} \in \mathbb{R}^m$ and a dataset of N data points $(\boldsymbol{X}, \boldsymbol{Y}) = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{y}_i \in \{1, ..., K\}$. The problem of *unstructured* neural network pruning can be formalized as finding a binary mask $\boldsymbol{M} \in \{0, 1\}^m$ that optimizes the following objective

$$\min_{\boldsymbol{M}} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\boldsymbol{f}(\boldsymbol{x}_i; \mathcal{A}(\boldsymbol{\theta}_0, \boldsymbol{M}) \odot \boldsymbol{M}), \boldsymbol{y}_i) \qquad (1)$$
$$\text{s.t. } \boldsymbol{M} \in \{0, 1\}^m, \quad \|\boldsymbol{M}\|_0/m \leq 1 - q,$$

where $\mathcal{L}$ is a suitable loss function for the downstream task, q is the desired sparsity of the resulting subnetwork and $\boldsymbol{\theta}_0$ are the initial parameters. $\mathcal{A}$ is an optimization algorithm (*e.g.* SGD [39], Adam [24]) that takes as input the mask $\boldsymbol{M}$ and the initialization $\boldsymbol{\theta}_0$ and returns the trained parameters at convergence $\boldsymbol{\theta}_{final} \odot \boldsymbol{M}$, where $\odot$ denotes the element-wise (Hadamard) product.

Due to the practical intractability of the described optimization problem, recent PaI algorithms focus on the notion of *saliency*, which is used as a score to assess the significance of network parameters regarding some property F of the network. After ranking the parameters' scores, only the top-S mask elements are retained and the final mask is used to approximate a solution for Eq. (1). Formally, the saliency takes the following form

$$S(\boldsymbol{\theta}) = \frac{\partial F}{\partial \boldsymbol{\theta}} \odot \boldsymbol{\theta} . \qquad (2)$$

For instance, in SNIP [30] the saliency is the loss function: $F = \mathcal{L}(\boldsymbol{\theta} \odot \boldsymbol{M}; (\boldsymbol{X}, \boldsymbol{Y}))$. Thus, that method assigns to each parameter a score which reflects how the loss would change when removing that specific parameter from the network.

In this work, our goal is to devise a suitable saliency score that correctly reflects how much each weight contributes to the trace of the NTK. As described in detail in the next subsection, the NTK approximates the training dynamics of the network [29], so removing those weights that minimally change its trace will result in small variations in the NTK spectrum, producing a subnetwork with similar predictive potential to the original larger network.

3.2. Neural Tangent Kernel and Pruning

We indicate the output of a ReLU-based neural network as $\boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta}) \in \mathbb{R}^{NK}$. Under the *gradient flow* regime (*i.e.* continuous-time gradient descent, with learning rate α) we can use a first-order Taylor expansion to approximate the network's output at a time step t of the optimization process:

$$\boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta}_{t+1}) = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta}_t) - \alpha \boldsymbol{\Theta}_t(\boldsymbol{X}, \boldsymbol{X}) \nabla_{\boldsymbol{f}} \mathcal{L} . \qquad (3)$$

The matrix $\boldsymbol{\Theta}_t(\boldsymbol{X}, \boldsymbol{X}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta}_t) \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta}_t)^T \in \mathbb{R}^{NK \times NK}$ is the *Neural Tangent Kernel* at time step t [23]. For infinitely wide networks, the exact training dynamics is described by the NTK which is a constant matrix that depends only on data, initialization, and architecture. It holds $\boldsymbol{\Theta}_t(\boldsymbol{X}, \boldsymbol{X}) = \boldsymbol{\Theta}_0(\boldsymbol{X}, \boldsymbol{X})$, thus we can drop the subscripts. Further works [29] observed that the NTK can approximate the training dynamics of networks of any depth without necessarily being infinitely wide by rendering its theory usable in practice. Additionally, it has been shown that a faster convergence is correlated with the direction in the parameter space pointed by the eigenvector with the largest corresponding eigenvalue of the NTK [2].

It is apparent that the NTK and its spectrum encapsulate crucial information about their model and offer an appealing way to evaluate the alignment between two networks. Models sharing the same NTK exhibit similar training dynamics [49], even with different parameter counts. Empirical results indicate that sparse subnetworks maintaining the NTK's largest eigenvalues of their dense counterpart converge more rapidly [48, 49] and better replicate the training dynamics of denser networks [26, 43].

While the strategy of using the NTK for network pruning seems promising, calculating the entire NTK spectrum is only feasible for very small neural networks with limited data [2]. For context, recent results on the NTK computation state a time complexity of $N^2 K[FP]$ [34], where N is the size of the dataset, K is the output size of the network, and $[FP]$ is the cost of a single forward pass. Indeed previous pruning methods that exploited the NTK theory either resorted to different approximations of the NTK spec-

trum [15, 37, 49] or indirectly tried to preserve it by maintaining the gradient flow in the network [48].

The next subsection explains how, by analyzing the activation paths within a neural network, it is possible to obtain an analytical decomposition of the trace of the NTK that is instrumental for PaI.

3.3. Neural Tangent Kernel and Activation Paths

Let $\mathcal{P}$ be the set of all paths connecting any input neuron to any output neuron of the network $\boldsymbol{f}$, where the edges of those paths are the weights[1] of the network. Each specific path can be referred to by its index $p = 1, ..., P$ in the set $\mathcal{P}$. The presence of weight θ_i in path p is denoted as $p_i = \mathbb{I}[\theta_i \in p]$. We can now define the product of weights within a path p as $v_p(\boldsymbol{\theta}) = \prod_{i=1}^{m} \theta_i^{p_i}$, where p_i is the exponent. Given an input example $\boldsymbol{x} \in \boldsymbol{X}$, the activation status of a path is $a_p(\boldsymbol{x}, \boldsymbol{\theta}) = \prod_{\{i|\theta_i \in p\}} \mathbb{I}[z_i > 0]$, where z_i is the activation of the neuron connected to the previous layer through θ_i. Thus, we can describe the k-th component of the output function of the network as:

$$\boldsymbol{f}^k(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{s=1}^{d} \sum_{p \in \mathcal{P}_{s \to k}} \boldsymbol{v}_p(\boldsymbol{\theta}) a_p(\boldsymbol{x}, \boldsymbol{\theta}) \boldsymbol{x}_s , \quad (4)$$

where $\boldsymbol{x}_s$ indicates the s-th term of the $\boldsymbol{x}$ vector and $\mathcal{P}_{s \to k}$ is the set of all paths from the input s to output neuron k [33].

By applying the chain rule it is possible to factorize the NTK as follows [15]:

$$\begin{aligned}
\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{X}) &= \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \theta) \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \theta)^T \\
&= \frac{\partial \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^T \\
&= \frac{\partial \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta})}{\partial \boldsymbol{v}(\boldsymbol{\theta})} \frac{\partial \boldsymbol{v}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \frac{\partial \boldsymbol{v}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^T \frac{\partial \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta})}{\partial \boldsymbol{v}(\boldsymbol{\theta})}^T \\
&= \boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X}) \boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}} (\boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}})^T (\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X}))^T \\
&= \boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X}) \boldsymbol{\Pi}_{\boldsymbol{\theta}} (\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X}))^T .
\end{aligned} \quad (5)$$

Here $\boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}} \in \mathbb{R}^{P \times m}$ compose the so-called *Path Kernel* matrix $\boldsymbol{\Pi}_{\boldsymbol{\theta}} \in \mathbb{R}^{P \times P}$ which is symmetric positive semi-definite and depends solely on the initialization and the network's architecture. The eigenvectors of the Path Kernel can be described as a collection of paths where the eigenvector associated with the largest eigenvalue represents the set of paths that maximize the flow within the network [15]. On the other hand, the matrix $\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X}) \in \mathbb{R}^{NK \times P}$, which we renamed, *Path Activation Matrix*, represents the change in output with respect to path values and entirely captures the dependence of $\boldsymbol{f}$ on the inputs by reweighting the paths within the network based on the training data.

---

[1]We use the terms "parameters" and "weights" interchangeably to refer to the network's parameters $\boldsymbol{\theta}$, as paths within a neural network are weighted by the value of each parameter.

Considering the eigenvalues π_i, ν_i and λ_i respectively of $\boldsymbol{\Pi}_{\boldsymbol{\theta}}$, $\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X})$ and $\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{X})$, it was demonstrated that $Tr[\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{X})] = \sum_i^{NK} \lambda_i \leq \sum_i^{NK} \nu_i \pi_i$ [15]. Previous works mentioning this upper bound [15, 37], end up focusing only on the Path Kernel of the pruned networks and maximize its trace $Tr(\boldsymbol{\Pi}_{\boldsymbol{\theta}}) = \sum_i \pi_i$ to preserve the largest NTK eigenvalue of the original network, which produces the highest flow through the network and hence, similar training dynamics. However, this might be misleading as the data-dependent term $\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X})$ is neglected.

In the following, we present a new upper bound for the NTK's trace that considers both the Path Kernel and the Path Activation Matrix, along with an exact calculation method, forming the core of our novel Path eXclusion approach for pruning.

3.4. Foresight Pruning via Path eXclusion

Starting from the decomposition of the NTK in Eq. (5) and from the definition of the Frobenius norm $\|A\|_F = \sqrt{Tr(AA^T)}$ it is possible to write

$$\begin{aligned}
Tr[\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{X})] &= Tr[\nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta})^T] \\
&= \|\nabla_{\boldsymbol{\theta}} \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{\theta})\|_F^2 \\
&= \|\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X}) \boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}}\|_F^2 \\
&\leq \|\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X})\|_F^2 \cdot \|\boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}}\|_F^2 ,
\end{aligned} \quad (6)$$

where the last inequality arises from the submultiplicative property of the matrix norm. It is easy to show that

$$\begin{aligned}
\|\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X})\|_F^2 &= \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{p=1}^{P} \left(\sum_{s=1}^{d} \mathbb{I}[p \in \mathcal{P}_{s \to k}] a_p(\boldsymbol{x}_n, \boldsymbol{\theta}) \boldsymbol{x}_{n_s} \right)^2 \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{p=1}^{P} a_p(\boldsymbol{x}_n, \boldsymbol{\theta}) \boldsymbol{x}_{n_{s|s \in p}}^2 ,
\end{aligned} \quad (7)$$

where $\boldsymbol{x}_{n_s}$ is the s-th component of the n-th sample vector $\boldsymbol{x}$. This term captures the dependence of the NTK's trace on the input data by choosing which paths are active and re-weighting by the input activations. The second term of the upper bound relates to the Path Kernel and as already discussed in [15], it holds

$$\|\boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}}\|_F^2 = \sum_{p=1}^{P} \sum_{j=1}^{m} \left(\frac{\boldsymbol{v}_p(\boldsymbol{\theta})}{\boldsymbol{\theta}_j} \right)^2 . \quad (8)$$

Both Eq. (7) and (8) can be calculated efficiently by exploiting the implicit computation of the network's gradients via automatic differentiation. To do that we introduce two auxiliary networks $\boldsymbol{h}$ and $\boldsymbol{g}$ which have the same architecture as the original $\boldsymbol{f}$ and are described by the input data $\boldsymbol{x}$, their parameters $\boldsymbol{\theta}$, and the status $\boldsymbol{a}$ of their ReLU activations.

Using $\mathbb{1}$ to indicate a vector of 1's, the k-th component of the output for each of these networks is defined as

$$\boldsymbol{h}^k(\mathbb{1}, \boldsymbol{\theta}^2, \mathbb{1}) = \sum_{p=1}^{P} \boldsymbol{v}_p(\boldsymbol{\theta}^2) = \sum_{p=1}^{P} \boldsymbol{v}_p^2(\boldsymbol{\theta}) \;,$$

$$\boldsymbol{g}^k(\boldsymbol{x}^2, \mathbb{1}, \boldsymbol{a}) = \sum_{p=1}^{P} a_p(\boldsymbol{x}, \boldsymbol{\theta}) \boldsymbol{x}_{s|s\in p}^2 \;.$$

Here $\boldsymbol{h}$ takes simplified data $\mathbb{1}$ as input, with squared parameters, and a vector of activations that are all one. Instead $\boldsymbol{g}$ takes the squared data as input, the parameters are all one, and the activations status is an exact copy of that of $\boldsymbol{f}$. Finally, we consider these two networks working jointly with the overall behavior described by

$$\mathcal{R}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{a}) = \sum_{n=1}^{N}\sum_{k=1}^{K} \boldsymbol{g}^k(\boldsymbol{x}_n^2, \mathbb{1}, \boldsymbol{a}_n) \cdot \boldsymbol{h}^k(\mathbb{1}, \boldsymbol{\theta}^2, \mathbb{1}) \;.$$

We can compute the gradient of this function via backpropagation obtaining

$$\sum_{j=1}^{m} \frac{\partial \mathcal{R}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{a})}{\partial \boldsymbol{\theta}_j^2} =$$

$$= \frac{\partial}{\partial \boldsymbol{\theta}_j^2} \sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{j=1}^{m} \boldsymbol{g}^k(\boldsymbol{x}_n^2, \mathbb{1}, \boldsymbol{a}_n) \cdot \boldsymbol{h}^k(\mathbb{1}, \boldsymbol{\theta}^2, \mathbb{1})$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} \boldsymbol{g}^k(\boldsymbol{x}_n^2, \mathbb{1}, \boldsymbol{a}_n) \cdot \sum_{j=1}^{m} \frac{\partial \boldsymbol{h}^k(\mathbb{1}, \boldsymbol{\theta}^2, \mathbb{1})}{\partial \boldsymbol{\theta}_j^2}$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{p=1}^{P} a_p(\boldsymbol{x}_n, \boldsymbol{\theta}) \boldsymbol{x}_{n_{s|s\in p}}^2 \cdot \sum_{p=1}^{P}\sum_{j=1}^{m} \frac{\boldsymbol{v}_p^2(\boldsymbol{\theta})}{\boldsymbol{\theta}_j^2}$$

$$= \|\boldsymbol{J}_{\boldsymbol{v}}^{\boldsymbol{f}}(\boldsymbol{X})\|_F^2 \cdot \|\boldsymbol{J}_{\boldsymbol{\theta}}^{\boldsymbol{v}}\|_F^2 \;.$$

Thus we are able to explicitly compute the upper bound in Eq. (6) and the value of each of the m components of the gradient $\partial \mathcal{R}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{a})/\partial \boldsymbol{\theta}_j^2$ is our saliency score indicating the importance of each parameter $\boldsymbol{\theta}_j$ in composing the trace of the NTK. To summarize, the final PX saliency score is:

$$S_{\text{PX}}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{a}) = \frac{\partial \mathcal{R}(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{a})}{\partial \boldsymbol{\theta}^2} \odot \boldsymbol{\theta}^2. \qquad (9)$$

As we perform global masking, we can observe from Eq. (7), (8) and (9) that our saliency function yields only positive scores which means that the saliency among layers is conserved. Combined with the iterative application of our pruning procedure, we satisfy the hypotheses of the Theorem of Maximal Critical Compression [44], which allows us to avoid *layer collapse*, namely pruning all neurons within one layer and preventing the information flow.

The full PX algorithm pseudocode is provided in the supplementary material.

4. Experiments

In this section, we describe the results of our experimental analysis that thoroughly compares our PX with several baseline methods. In terms of datasets, tasks, and architectures we align with the literature and adopt well-established setups that are briefly summarized in the following [6, 13, 14, 30, 44]. Moreover, we investigate whether PaI can be applied to pre-trained models without damaging their downstream transferability. Our empirical evaluation provides a positive answer to this new research question.

Datasets & Tasks. For the classification experiments, we use CIFAR-10, CIFAR-100 [27], Tiny-ImageNet and ImageNet [10]. For the segmentation experiments we follow [6] and use the training and validation splits of Pascal VOC 2012 [12] for model learning and evaluation.

Architectures. As done by [44, 49], for the classification experiments we use ResNet-20 [20] on CIFAR-10, VGG-16 [40] on CIFAR-100. ResNet-18 on Tiny-ImageNet and ResNet50 on the ImageNet dataset. By following [6], on the segmentation task we use DeepLabV3+ [5] with ResNet-50.

Initialization. As in [14] we initialize each model using Kaiming normal initialization [19]. Furthermore, we assess how pre-trained parameters affect the foresight pruning procedure. For this analysis we adopt a setting analogous to that in [6] that originally considered only iterative unstructured magnitude pruning. Specifically, we use a ResNet-50 pre-trained on ImageNet as well as two self-supervised models obtained with MoCov2 [7] and CLIP [38].

Implementation details. Regarding the training procedure we follow [14] and [6] when assessing respectively our PX with respect to the PaI state-of-the-art methods and the pre-training transferability. We evaluate each algorithm on trivial (36.00%, 59.04%, 73.80%), mild (83.22%, 89.30%, 93.12%) and extreme (95.60%, 97.17%, 98.20%) sparsity ratios as [49]. We use 100 rounds for iterative PaI methods adopting an exponential schedule as [14, 44]. Full implementation details can be found in the supplementary.

4.1. Classification with Random Initialization

To provide an empirical evaluation of the strengths and effectiveness of our method, we compare PX with state-of-the-art foresight pruning algorithms. These include both data-driven methods like SNIP [30] and GraSP [48], as well as data-agnostic techniques such as SynFlow [44] and NTK-SAP [49]. We also include two common baselines in PaI which are Random pruning and Magnitude-based pruning.

CIFAR-10, CIFAR-100 & Tiny-ImageNet. In Fig. 2. we report the classification results when using ResNet-20, VGG-16 and ResNet-18 on CIFAR-10, CIFAR-100 and Tiny-Imagenet, respectively. For low sparsity levels, most techniques exhibit strong performance. On ResNet-20 (CIFAR-10), the performance gap across techniques is
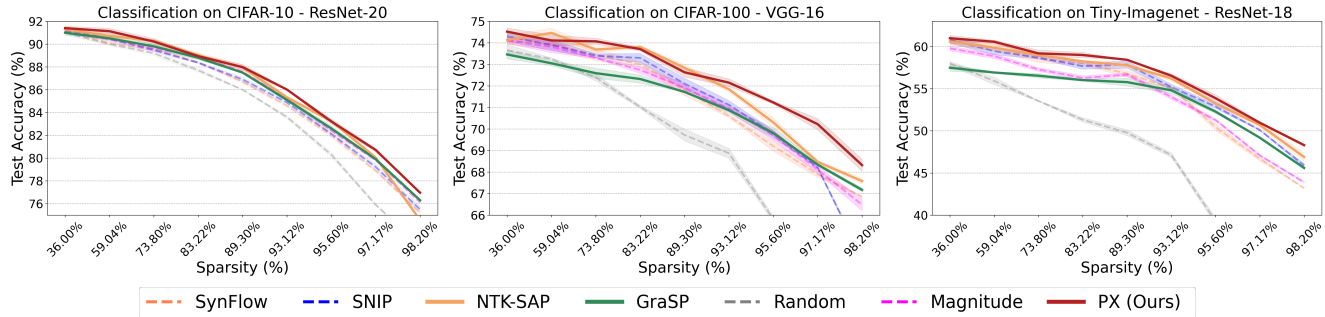
Figure 2. Average classification accuracy at different sparsity levels on CIFAR-10 using ResNet-20, CIFAR-100 using VGG-16 and Tiny-ImageNet using ResNet-18, respectively. Each experiment is repeated three times. We report in shaded colors the standard deviation.

low, albeit being slightly higher for random pruning and GraSP. On VGG-16 (CIFAR-100) and ResNet-18 (Tiny-ImageNet), a comparable pattern emerges, but the disparity in performance widens noticeably. In all three cases, our approach consistently delivers the highest results. At moderate sparsity levels, PX and NTK-SAP start to emerge as the top performers across all three experiments. Specifically, on VGG-16 (CIFAR-100), these two methods showcase competitive performance, with NTK-SAP being slightly ahead. The ranking of the other techniques remains consistent at these levels. Finally, at extreme levels of sparsity, PX clearly outperforms all the competitors. In particular, on VGG-16 (CIFAR-100), there is a considerable disparity between PX and all other methods, including NTK-SAP. Notably, there's a substantial decline in NTK-SAP's performance at 98.20% sparsity on ResNet-20 (CIFAR-10). Across all three experiments, GraSP, despite initially exhibiting lower performance, demonstrates commendable consistency in maintaining its results.

ImageNet. We conducted a comprehensive assessment of PX on larger-scale datasets, specifically ImageNet, employing ResNet-50 as the backbone model. In line with [49], we examined two sparsity levels (89.26% and 95.60%). Our findings, detailed in Table 1., reveal that PX, NTK-SAP, and GraSP get top results, with PX exhibiting a slight advantage. Magnitude pruning surprisingly demonstrates greater competitiveness compared to SNIP, which performed well on smaller-scale datasets but ranks last in this evaluation.

4.2. Starting From Pre-Trained Parameters

Within this section, we examine the impact of initialization from pre-trained models: we aim at gaining insights into how we can leverage PaI algorithms to efficiently transfer knowledge to downstream tasks.

Classification. Fig. 3. presents our findings on classification results across CIFAR-10, CIFAR-100, and Tiny-ImageNet using ResNet-50 initialized from ImageNet, Mo-Cov2 on ImageNet, and CLIP pre-trainings. We do not report SynFlow here as it produces exploding gradients while

Classification on ImageNet - ResNet-50		
Pruning Method	89.26%	95.60%
SynFlow [44]	66.48 ± 0.12	59.41 ± 0.19
SNIP [30]	60.50 ± 0.34	45.82 ± 0.35
NTK-SAP [49]	67.98 ± 0.31	59.84 ± 0.30
GraSP [48]	67.21 ± 0.52	60.01 ± 0.16
Random	64.97 ± 0.27	56.79 ± 0.44
Magnitude	66.56 ± 0.23	47.80 ± 0.21
PX (Ours)	**68.11** ± 0.29	**60.28** ± 0.32

Table 1. Average classification accuracy at different sparsity ratios on the ImageNet dataset, using Kaiming normal initialized ResNet-50 as backbone. Each experiment is repeated three times. We report also the standard deviation. **Bold** indicates the best result. Underline the second best.

estimating saliency scores. This issue has been also observed in contexts like Neural Architecture Search [4].

Until reaching extreme sparsity, most methods closely align with the performance of the dense baseline, underlining that employing PaI in this context serves as a viable, cost-free alternative to Iterative Magnitude Pruning (IMP). At extreme sparsity levels across all experiments, PX consistently outperforms other methods maintaining proximity to the dense baseline on simpler tasks (CIFAR-10) and showing a clear advantage over the competitors on more complex tasks (CIFAR-100 and TinyImageNet).

We remark that, while NTK-SAP stands as a state-of-the-art method, its performance drastically diminishes at extreme sparsity levels when initialized from pre-trained parameters. We attribute this decline to the interference of random mini-batches with the batch normalization statistics of the pre-trained model during its saliency estimation. This provides clear evidence of the limitations of data-free PaI methods.

Segmentation. In Fig. 4., we present the semantic segmentation results on the Pascal VOC2012 datasets, employing DeepLabV3+ on ResNet-50 initialized with Mo-Cov2 on ImageNet (further results with other initialization in the supplementary). Here we report only the results for our method, SNIP, Random, and Magnitude-based pruning. This selective reporting stems from issues encountered with
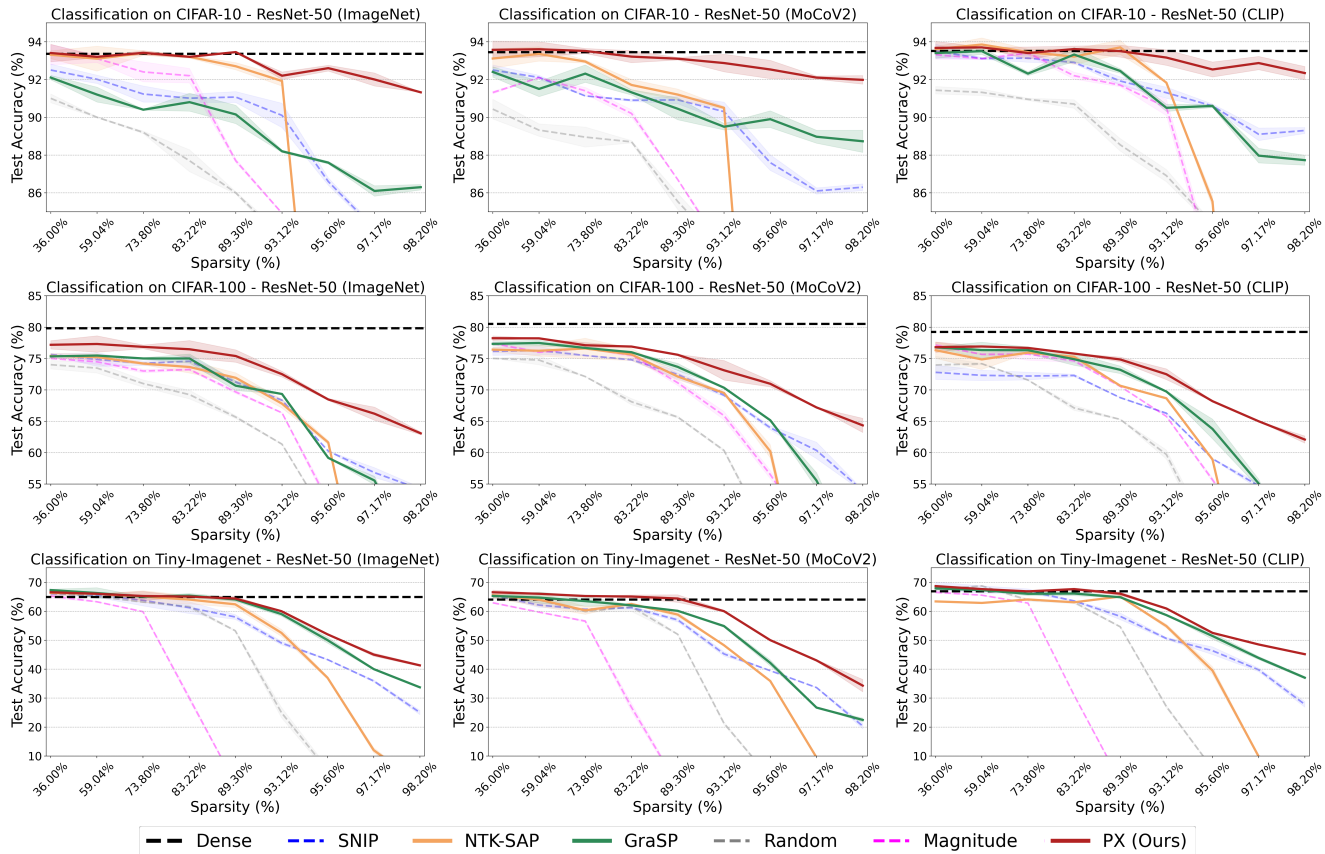
Figure 3. Average classification accuracy at different sparsity levels on CIFAR-10, CIFAR-100 and Tiny-ImageNet using pre-trained ResNet-50 as architecture. The first column reports the results of starting from the supervised ImageNet pre-training. The second column reports the performance when starting from the MoCov2 pre-training on ImageNet. Finally, in the third column we report the results when starting from CLIP. Each experiment is repeated three times. We report in shaded colors the standard deviation.
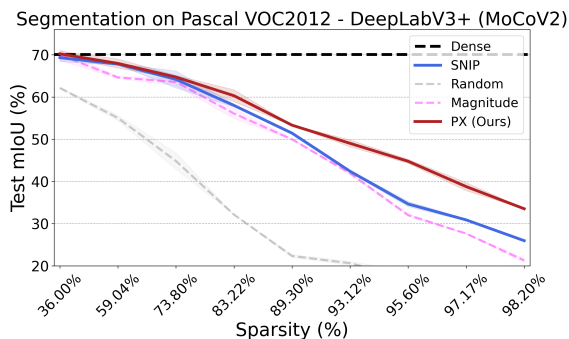
Figure 4. Average mean Intersection over Union (mIoU) at different sparsity levels on Pascal VOC2012 using DeepLabV3+ with pre-trained ResNet-50 as the backbone. Each experiment is repeated three times. Standard deviations are in shaded colors.

other methodologies: SynFlow faced again challenges with exploding gradients, NTK-SAP resulted in layer collapse within the segmentation head due to the potential absence of positive saliency scores, a crucial factor in preventing such collapses. Similarly, GraSP, relying on a single round of pruning, encountered limitations in its applicability.

PX confirms its superiority to the other methods even for semantic segmentation. We also note that SNIP consistently demonstrates good performance and it appears as a remarkable result in comparison to the failure of NTK-SAP. Overall, despite the encouraging results, it is apparent that significant effort needs to be directed toward PaI techniques specifically tailored for more complex vision tasks such as semantic segmentation. As of now, PaI methods only approximate the dense network results at trivial sparsity levels. Surprisingly, Magnitude-based pruning also finds competitive subnetworks, comparable to other PaI methods, but only at trivial and moderate sparsity levels.

4.3. Spectral Analysis of the Fixed-Weight-NTK

We designed PX to preserve the parameters that mostly influence the eigenspectrum of the NTK. To verify this behavior we visualize in Fig. 5. the eigenvalues distributions of the Fixed-Weight-NTK[2]. Our investigation focuses on pruned ResNet-20 subnetworks on CIFAR-10, at 93.12%

---

[2]Note that in the limit of infinite width the Fixed-Weight-NTK [49] approaches the Analytic NTK [23].
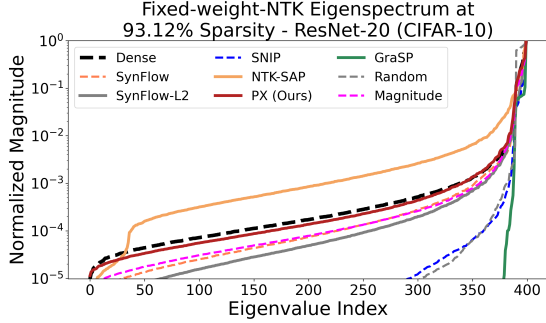
Figure 5. Fixed-Weight-NTK spectrum of ResNet-20 on the CIFAR-10 dataset at 93.12% sparsity ratio.
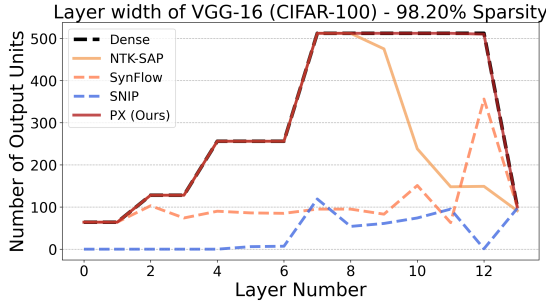

Figure 6. Active output units at 98.20% sparsity in VGG-16. For SNIP and PX data mini-batches are sampled from CIFAR-100.

sparsity. Notably, even at this substantial level of sparsity, our approach closely mirrors the eigenspectrum of the original dense network confirming the expectations. To make our argument even more solid, we disregarded the data-dependent term in PX which implies falling back to SynFlow-L2. As can be observed, data play a central role when preserving the eigenspectrum of the NTK. This ablation study further reinforces our claims.

4.4. Layer Width

As discussed earlier, the integration of iterative pruning rounds and the layer-wise preservation of saliency are pivotal in preventing layer collapse using saliency functions [44]. However, [37] observed that iterative PaI methods suffer a sudden reduction in the number of output units, leading to narrow layers and bottlenecks. Recent investigations [16] highlighted a correlation between network performance and higher output width when fixing a certain number of parameters in an architecture. Motivated by this, we investigate more in-depth the behavior of PX.

For fully connected layers every neuron constitutes an output unit. For convolutional layers, we follow [37] and consider each kernel as an output unit. If all parameters within a convolutional kernel are pruned, the layer's output unit count is reduced by one. In Figure 6, we conduct the analysis on the number of output units that each iterative PaI method preserves after pruning. Despite being iterative, PX is not affected by the issues mentioned in [37] and consistently preserves the output width of each layer.

	T = 1		T = 2		T = 10		T = 100		T = 200		Epochs = 960	
	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time
IMP	-	-	-	-	-	-	-	-	-	-	77.38	13708.80
Single-shot PaI methods												
Random	72.31	0.0003	-	-	-	-	-	-	-	-	-	-
Magnitude	76.12	0.0006	-	-	-	-	-	-	-	-	-	-
SNIP [30]	75.39	0.17	-	-	-	-	-	-	-	-	-	-
GraSP [48]	76.30	0.39	-	-	-	-	-	-	-	-	-	-
Iterative PaI methods												
SynFlow [44]	10.00	0.13	71.20	0.27	73.98	1.34	75.19	13.40	75.67	26.81	-	-
NTK-SAP [49]	18.85	0.21	55.87	0.42	72.21	2.09	74.55	20.92	75.06	41.85	-	-
PX (Ours)	76.15	0.14	76.47	0.28	77.04	1.41	77.08	14.13	77.39	28.27	-	-

ResNet-20 (CIFAR-10) - 98.20% Sparsity

Table 2. Comparison of the accuracy and execution time (in seconds) when varying the number of pruning rounds T, starting from ResNet-20 on CIFAR-10. The timings reported refer to the 95th percentile of 100 measurements.

4.5. Execution Time Analysis

In Table 2, we show the effect of changing the number of pruning rounds T, presenting accuracy and total execution time of the pruning procedures in seconds. Here we include IMP to offer a broader context to our study. We ran our evaluation on NVIDIA Titan Xp GPU, Intel i7-9800X CPU, and using the `perf_counter` clock from Python's `time` module. IMP ($Epochs = 960$) takes nearly 4 hours to outperform PX ($T = 10$). GraSP leads at $T = 1$, but PX surpasses GraSP at $T = 2$ without exceeding its time cost. Increasing T beyond 100 marginally improves results but does not alter conclusions, in line with [44, 49]. More discussions about the computational cost of PX and its competitors are provided in the supplementary.

5. Conclusion

Pruning at initialization offers the attractive possibility of reducing the number of parameters in a neural network, bypassing the need for training to identify the pruning mask. The NTK and its pathwise decomposition provide a powerful proxy for identifying parameters that are important for preserving training dynamics after pruning. While most methods consider only the data-independent component, we propose a new upper bound on the trace of the NTK which led to Path eXclusion (PX), that allows us to preserve its spectrum and consider the data-dependent component as well. We show experimentally that PX is not only robust to different architectures and tasks but can also be effectively used to search for subnetworks in large pre-trained models that retain almost intact transferability.

References

[1] Milad Alizadeh, Shyam A Tailor, Luisa M Zintgraf, Joost van Amersfoort, Sebastian Farquhar, Nicholas Donald Lane, and Yarin Gal. Prospect pruning: Finding trainable weights at initialization using meta-gradients. In *ICLR*, 2022. 1, 2

[2] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *ICML*, 2019. 3

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Sub-biah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 2020. 1

[4] Niccolò Cavagnero, Luca Robbiano, Barbara Caputo, and Giuseppe Averta. Freerea: Training-free evolution-based architecture search. In *WACV*, 2023. 2, 6

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 5

[6] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *CVPR*, 2021. 2, 5

[7] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020. 5

[8] Pau De Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In *ICLR*, 2021. 2

[9] Pau De Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In *ICLR*, 2021. 2

[10] 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. 5

[11] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *CVPR*, 2020. 1

[12] Mark Everingham, Ali SM Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015. 5

[13] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 1, 2, 5

[14] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *ICLR*, 2021. 3, 5

[15] Thomas Gebhart, Udit Saxena, and Paul Schrater. A unified paths perspective for pruning at initialization. *arXiv preprint arXiv:2101.10552*, 2021. 2, 4

[16] Anna Golubeva, Behnam Neyshabur, and Guy Gur-Ari. Are wider nets better given the same number of parameters? In *ICLR*, 2021. 8

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015. 1, 2

[18] B. Hassibi, D.G. Stork, and G.J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE ICNN*, 1993. 2

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 5

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(1), 2021. 3

[22] Eugenia Iofinova, Alexandra Peste, Mark Kurtz, and Dan Alistarh. How well do sparse imagenet models transfer? In *CVPR*, 2022. 2

[23] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018. 2, 3, 7

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 3

[25] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. In *ICCV*, 2023. 1

[26] Dmitry Kopitkov and Vadim Indelman. Neural spectrum alignment: Empirical study. In *ICANN*, 2020. 3

[27] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009. 5

[28] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In *NeurIPS*, 1989. 2

[29] 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

[30] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019. 1, 2, 3, 5, 6, 8

[31] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018. 1, 2

[32] Alexandra Sasha Luccioni and Alex Hernandez-Garcia. Counting carbon: A survey of factors influencing the emissions of machine learning. *arXiv preprint arXiv:2302.08476*, 2023. 1

[33] Qi Meng, Shuxin Zheng, Huishuai Zhang, Wei Chen, Zhi-Ming Ma, and Tie-Yan Liu. G-sgd: Optimizing relu neural networks in its positively scale-invariant space. In *ICLR*, 2019. 4

[34] Roman Novak, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Fast finite width neural tangent kernel. In *ICML*, 2022. 3

[35] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *NeurIPS*, 2015. 2

[36] NVIDIA. cusparse. https://developer.nvidia.com/cusparse, 2023. 1

[37] Shreyas Malakarjun Patil and Constantine Dovrolis. Phew: Constructing sparse networks that learn fast and generalize well without training data. In *ICML*, 2021. 2, 4, 8

[38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 1, 5

[39] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 3

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[41] Suraj Srinivas and R. Venkatesh Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016. 2

[42] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *ACL*, 2019. 1

[43] Lili Su and Pengkun Yang. On learning over-parameterized neural networks: A functional approximation perspective. In *NeurIPS*, 2019. 3

[44] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020. 1, 2, 3, 5, 6, 8

[45] Nigel Toon. Introducing 2nd generation ipu sytems for ai at scale. https://www.graphcore.ai/posts/introducing-second-generation-ipu-systems-for-ai-at-scale, 2020. 1

[46] Sagar Verma and Jean-Christophe Pesquet. Sparsifying networks via subdifferential inclusion. In *ICML*, 2021. 1, 2

[47] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In *ICML*, 2019. 1, 2

[48] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020. 2, 3, 4, 5, 6, 8

[49] Yite Wang, Dawei Li, and Ruoyu Sun. Ntk-sap: Improving neural network pruning by aligning training dynamics. In *ICLR*, 2023. 2, 3, 4, 5, 6, 7, 8

[50] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NeurIPS*, 2016. 1, 2

[51] Yinan Yang, Yu Wang, Ying Ji, Heng Qi, and Jien Kato. One-shot network pruning at initialization with discriminative image patches. In *BMVC*, 2022. 3