When to Prune? A Policy towards Early Structural Pruning

Maying Shen, Pavlo Molchanov, Hongxu Yin, Jose M. Alvarez
NVIDIA
{mshen, pmolchanov, dannyy, josea}@nvidia.com

Abstract

Pruning enables appealing reductions in network memory footprint and time complexity. Conventional post-training pruning techniques lean towards efficient inference while overlooking the heavy computation for training. Recent exploration of pre-training pruning at initialization hints on training cost reduction via pruning, but suffers noticeable performance degradation. We attempt to combine the benefits of both directions and propose a policy that prunes as early as possible during training without hurting performance. Instead of pruning at initialization, our method exploits initial dense training for few epochs to quickly guide the architecture, while constantly evaluating dominant sub-networks via neuron importance ranking. This unveils dominant sub-networks whose structures turn stable, allowing conventional pruning to be pushed earlier into the training. To do this early, we further introduce an Early Pruning Indicator (EPI) that relies on sub-network architectural similarity and quickly triggers pruning when the sub-network's architecture stabilizes. Through extensive experiments on ImageNet, we show that EPI empowers a quick tracking of early training epochs suitable for pruning, offering same efficacy as an otherwise "oracle" grid-search that scans through epochs and requires orders of magnitude more compute. Our method yields 1.4% *top-1 accuracy boost over state-of-the-art pruning counterparts, cuts down training cost on GPU by* $2.4\times$*, hence offers a new efficiency-accuracy boundary for network pruning during training.*

1. Introduction

The success of convolutional neural networks (CNNs) fuels the recent progress in computer vision, boosting up performance for classification, detection, and segmentation tasks [16, 35, 37]. While enjoying the accuracy benefits CNNs bring, a simultaneous increase in network complexity imposes higher memory footprint and computing power consumption, making deployment of CNNs on resource-constrained devices a challenging task [7, 29, 30]. In lieu of computation-intensive networks, recent work turn to compression techniques for efficient models leveraging pruning [3, 15, 25, 29], quantization [5, 43, 46], knowledge distillation [19, 31, 45], neural architecture search [38, 41, 44], and architecture redesigns [21, 28, 39]. Among these, pruning demonstrates to be a widely adopted method that compresses pre-trained models before deployment. The primary goal of pruning aims to remove insignificant network parameters without impacting accuracy. In particular, structural pruning removes entire filters (or neurons) as such the resulting structural sparsity benefits legacy off-the-shelf platforms, *e.g.*, CPUs, DSPs, and GPUs.

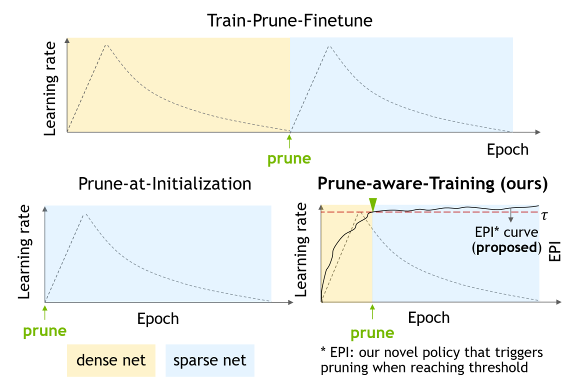


Figure 1. Pruning paradigm overview. ***Train-Prune-Finetune*** prunes after training, effective but costs additional training time; ***Prune-at-Initialization*** prunes right before training towards a smaller network, cuts down on training time but suffers notable performance degradation; ***Pruning-aware-Training*** **(ours)** prunes during training aiming at benefits from two worlds. It governs on post-pruning performance while aiming to minimize training time, via a new policy around Early Pruning Indicator (EPI) that signals an early optimal point to start pruning during training.

In general, network pruning involves three key steps: (i) original training of a dense model for high accuracy, (ii) pruning away insignificant weights to remove redundancy, and finally (iii) fine-tuning the pruned model to recover performance [15, 26, 30]. Despite remarkable compactness delivered by the last two steps, the original training of an over-parameterized network remains mostly untouched. Such approaches require a twice long time (resource) as an original training recipe given similar required computes for fine-tuning, making the entire pipeline slow, sometimes infeasible. For example, a very recent breakthrough in language modeling, a GPT-3 model [4], requires millions of dollars (more than 300 NVIDIA V100 GPU years) just for the initial training. Already aware of post-training redundancy, an interesting question arises - can we somehow prune a network during its initial training, as such the resulting sparsity can (i) immediately benefit training and (ii) save us from the costly additional fine-tuning upon training ends?

One intuitive and ideal solution to this problem is pruning the network right at initialization even before training starts. The intriguing observation from the Lottery Ticket Hypothesis [9] hints potential to this task: it shows the (i) existence of small sub-models, identifiable via pruning, within a large dense model, that can (ii) be trained in isolation to achieve the same accuracy as its dense counterpart [9, 25]. This field has quickly evolved and recent approaches have enhanced policy for optimal sub-network at the initialization by preserving the loss or the gradient flow [8, 24, 42]. Despite rapid progress, the approaches of sub-network identification at the initialization remain challenging and still suffer noticeable accuracy loss [10, 12].

Instead of *zero* training, in this work, we showcase the benefits and practicability of pruning *early* during training. Doing so allows one to (i) save compute by training only pruned models most of the time, (ii) alleviate any extra fine-tuning by aligning the process with original training, while (iii) suppressing accuracy loss by moving slightly later into the training regime for pruning guidance. We name this approach **pruning-aware-training** (PaT). As shown in Fig. 1, unlike pruning-at-initialization, PaT takes full advantage of early-stage dense model training that is beneficial for rapid learning and optimal architecture exploration [1, 11], while aiming to identify the best sub-network as early as possible, rather than waiting till training ends as in conventional pruning.

The key of benefiting from the training efficiency of PaT and saving training time relies on finding an early yet eligible point during training to start pruning. Existing methods that perform pruning during training [3, 10, 27, 33] have shown the efficacy of this direction by reducing the turnaround time. However, in most cases a fixed initial interval for pruning is set heuristically, or post-training statistics are required. In this work we focus on understanding how the starting point of pruning can be set automatically.

We start by analyzing in depth the evolution of pruned architectures via performing trimming across all epochs rigorously and compare their suitability for pruning. Though laborious, this oracle estimate offers key insights on pruning during training. We observe an important property: agnostic of magnitude or gradient criterion, (i) pruning at early epochs results in different final architectures, but (ii) dominant architecture emerges within just a few epochs and stabilizes thereafter till training ends, allowing conventional pruning to be pushed earlier into the training.

Amid such property we further propose a novel metric, called Early Pruning Indicator (EPI), that estimates the structure similarity between networks resulted in pruning at consecutive epochs of the same base model. Given intrinsic access to model weights and gradients during training, EPI can be calculated very efficiently alongside initial training without bells and whistles, while helping avoid the otherwise lengthy grid search for starting epochs. Once the resulting pruning structure will not vary between epochs we argue and demonstrate it is safe to prune. As prior work [25] and we observe, structural pruning acts as an architecture search and tries to find the optimal number of neurons per layer. Therefore, we hypothesize that pruning can be performed as soon as the architecture of the dominant sub-network becomes stable. We demonstrate that the proposed metric works across varying network architectures, pruning ratios, delivering consistent reductions in training time.

Our main contributions are as follows:

- We propose a novel metric called Early Pruning Indicator (EPI) that indicates an early point to start pruning during training. Our metric enables training to benefit from sparsity, significantly reducing training resources with minor accuracy drop.
- We demonstrate that for structural pruning (output channel pruning), initial dense training fuels accuracy boosts. Augmented by EPI, our pruning-aware-training outperforms pruning-at-initialization alternatives by a large margin.
- We show that EPI is agnostic to the pruning method used by showing efficacy for both magnitude-based and gradient-based pruning, enabling a new state-of-the-art boundary for training speedup through in-situ pruning.

2. Related Work

Network pruning. Mainstream pruning methods can be divided into three categories depending on when pruning is performed: 1) train-prune-finetune, 2) prune at initialization, and 3) prune while training.

The first group, train-prune-finetune, performs pruning on a densely pre-trained network and then, fine-tune the resulting structure to recover the performance loss caused by pruning. There are many methods aiming at preserving the final accuracy [17, 29, 30] and minimize the output change of each layer [18, 26]. A key focus of these work resides in identifying redundant connections whose removal brings the least perturbation to the overall performance. While enabling plausible performance and improving efficiency at test time, the aforementioned approaches cannot yet bring any efficiency benefit to training. Quite in contrary, most recipes result in nearly doubled training time amid the requirement for lengthy fine-tuning.

Prune at initialization methods, backed by the Lottery Ticket Hypothesis [9], question the necessity of dense training for performance convergence [25]. A forerunner in this group is SNIP [24], an approach that identifies a trainable sub-network at initialization. Subsequently, other methods such as FORCE [8], GraSP [42], or SynFlow [40] have been proposed to improve performance. These methods make the training more efficient as they only train the sparse network. However, the reliability of pruning at initialization remains unsatisfactory facing inevitable performance gaps [10].

Prune while training methods rest in the middle by finding a trade-off between training efficiency and final accuracy. Literature falls under two streams towards this task: a) regularization-based methods that encourage sparsity during training [3, 13, 27], and b) sub-ticket selection methods via saliency that discard redundancy [2, 14, 17]. Our work belongs to the latter given its efficacy to quickly enforce a pruning ratio and ease-of-control during training. Under this realm, one line of work learns sub-networks during training [3,27,33]. Others, such as Frankle *et al*. [10], study the need of few training iterations before pruning in order to maximize the performance. These methods struggle to automatically identify the starting points at which pruning can be performed, while heavily relying on hand-crafted or post-training heuristics for decision making.

Network similarity. Our policy explores the network similarity of two sub-networks resulted from pruning. A comprehensive review of network similarity measures was presented in [23]. These methods aim at comparing the representation between two fully trained models with different initialization, hence are not applicable to in-training gauging for pruning where weights and dominant architectures are both changing. To get the structural similarity for pruning, we focus on the number of remaining neurons across layers in a network when comparing with another one. The difference between these skeletons using coefficients can be directly measured by Spearman's [36] and Kendall's tau [22] rank correlation. However, these rank correlation metrics take into account the specific ranking and, more importantly, they would rely on all neurons in the network. Thus, they provide the same value for all pruning ratios.

Of particular relevance to this work is [10] by Frankle *et al*. that proposes an approach to measure the instability of a network structure to understand pruning viability. One noticeable finding by this work shows that the best time to perform iterative magnitude pruning tends to be after some initial training. Interestingly it identifies a relationship between the model instability and the accuracy of the pruned network, though the instability measure proposed by the method can only be measured *after training is completed*, hence remains insufficient to directly signal when to start pruning during training. EarlyBERT [6], a work on

Algorithm 1 Iterative pruning within one epoch

1: For prune ratio α, schedule the number of neurons to prune per step for S steps via exponential scheduler [8], forming $\mathbf{m} \in \mathbb{R}^S$
2: **while** $|\mathcal{P}| \leq \alpha|\mathcal{F}|$ **do**
3: Average importance calculated by (2) or (3) over multiple mini-batches
4: $\mathcal{P}$ is the indices of $\mathbf{m}_i$ bottom-ranked neurons
5: $\mathbf{W}_{\mathcal{P}} \leftarrow 0$ ▷ *remove pruned neurons*
6: $\mathcal{R} = \mathcal{F} - \mathcal{P}$ ▷ *the remaining neurons*
7: Update $\mathbf{W}_{\mathcal{R}}$; $i \leftarrow i + 1$
8: **end while**

NLP models, also studies the network similarity to guide the pruning. However, it uses the mask Hamming distance to measure the sub-network similarity. This requires the full pruning mask to be fixed between sub-networks, selecting the exact same neurons, thus impairing the flexibility.

3. Method

We next elaborate our early pruning algorithm in details.

3.1. Objective Function

Consider a neural network with L layers, each layer specified by its weight $\mathbf{W}^l \in \mathbb{R}^{C_O^l \times C_I^l \times K^l \times K^l}$, K being the kernel size and C_I and C_O being the number of input and output channels/neurons, respectively. Altogether, these parameters form the parameter set $\mathbf{W} = \{\mathbf{W}^l\}_{l=1}^L$ for the network. Given a training set consisting of N input-output pairs $\{(x_i, y_i)\}_{i=1}^N$, learning the parameters of a network under filter sparsity constraints can be expressed as solving the following optimization problem:

$$\arg\min_{\mathbf{W}} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i, \mathbf{W})) + r(\mathbf{W}), \text{ s.t. } \frac{|\mathcal{P}|}{|\mathcal{F}|} \geq \alpha \tag{1}$$

where $\ell(\cdot)$ denotes the loss function that compares the network prediction to the ground-truth, $f(\cdot)$ encodes the network transformation, $r(\cdot)$ is a regularizer acting on the network parameters, and α is the target pruning ratio. $\mathcal{F} = \{\mathcal{F}^l\}_{l=1}^L$ represents the index set of all the neurons in the network. This index set can be divided into two disjoints sets $\mathcal{P} = \{\mathcal{P}^l\}_{l=1}^L$ and $\mathcal{R} = \{\mathcal{R}^l\}_{l=1}^L$, representing the index sets of the pruned and remaining neurons respectively. We have $\mathcal{F} = \mathcal{P} \cup \mathcal{R}$ and $\mathcal{P} \cap \mathcal{R} = \emptyset$. PaT involves three stages: dense training, network pruning, and sparse training. During initial dense training, the forward/backward passes are computed using all the filters in the network $f(x_i, \mathbf{W}_{\mathcal{F}})$. While during sparse training, the forward/backward passes only use the remaining neurons $f(x_i, \mathbf{W}_{\mathcal{R}})$. Akin to [29, 30], we follow the paradigm of iterative pruning that finishes within one epoch. More precisely, as shown in Algorithm 1, the process consists of the following two steps. First, at each training iteration, we compute the importance metric of each neuron according to a pruning criterion. Meanwhile we keep updating network weights as normal. Then, at each pruning step, we get the averaged importance score for all neurons, and then remove the neurons with the smallest importance values. Each pruning step is carried out after seeing multiple training batches, usually several hundreds of batches are more than sufficient [29]. Note that though containing several quick interactive steps, the entire pruning can be finished very quickly within one training epoch.

For comprehensiveness we consider two popular criteria from literature - both magnitude-based and gradient-based schemes for neuron importance ranking:

Magnitude-based criterion uses the l_2-norm of the neuron weights to measure the relevance of a neuron:

$$\mathcal{I}_n^l = ||\mathbf{W}_n^l||_2 / \sqrt{P^l}, \tag{2}$$

where $P^l = C_I^l \times K^l \times K^l$ denotes the number of parameters per filter at layer l, and n specifies the output neuron index within $\mathbf{W}^l$. Such normalization ensures its comparability for neurons from different layers with different sizes [3].

Gradient-based criterion considers the Taylor expansion of the loss change to approximate the importance of a neuron. Initially, Molchanov *et al*. [30] proposed to estimate importance as a magnitude of the gradient-activation product, and more recently, SNIP [24] and FORCE [8] extended the idea to a parameter level. Specifically, gradient-based criterion using Taylor expansion for neurons can be defined as:

$$\mathcal{I}_n^l = \left| \sum\nolimits_{w \in \mathbf{W}_n^l} g_w w \right|, \tag{3}$$

where g_w is the gradient of the weight w. The metric estimates an approximate change in the loss function once the neuron is removed. As suggested in [29], for networks using batch normalization, the best way to apply pruning is on the batch normalization layers instead of convolutional filters directly. Additionally, the loss of removing the channel can be approximated via accumulative effect of the learnable scale and shift: $\mathcal{I}_n^l = \left| g_{\gamma_n^l} \gamma_n^l + g_{\beta_n^l} \beta_n^l \right|$, where γ and β are the weight and bias of the batch normalization layer, respectively. We empirically observe slight improvements using L_1 for pruning during training rather than the original L_2 as in [29] for post-training pruning.

3.2. Towards Early Pruning

Recall that our goal is to maximize the accuracy of the network while minimizing the compute required for training. This compute is usually dominated by the amount of time performing dense training. The sooner we prune the network, the less resources it requires to finish training.

As both prior work [11] and we empirically observe, the early stage of neural network training imposes a rapid motion in parameter space with large gradient magnitudes. This generates fruitful information for initial network convergence and a quick accuracy boost. With such a fact, an intuitive option for early pruning can be to analyze the emergence of important neurons at different training stages that the network gradually picks up for the underlying task, and then prune the insignificant ones right away.

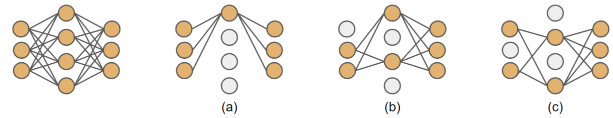


Figure 2. Structure of different sub-networks. Colored circles and solid lines are active neurons and connections. Sub-figures (a), (b) and (c) are three different sub-networks of the original network on the left. While these sub-networks having the same number of neurons in total, sub-networks (b) and (c) are in higher similarity.

Given its intrinsic access to weights and gradients, training allows one to quickly rank all the neurons globally with very little extra compute. This allows the network to take a quick glimpse into the problem from the architecture space that is empirically observed informative [16, 20, 37], while we can quite efficiently track architectural convergence.

To this end, we check after each training epoch for a sub-network specified by the top k most important neurons globally according to the chosen pruning criterion. Close to but different from a final winning ticket, an intermediate helps identify dominant neurons, but remains not as strong as its final version, while constantly changing during training. However, as we will show later, the architecture of such sub-network changes rapidly in the first few epochs, then surprisingly shows minimal changes thereafter for the remaining epochs. Knowingly exploiting such fast convergence to stability and slow changes of dominate sub-network thereafter, we argue and demonstrate pruning can be started as early as when its *Top-k* sub-network stabilizes. Next, we explore network similarity to signal such stability.

3.3. Early Pruning Indicator (EPI)

We look into the structural similarity between dominant sub-networks to quantify architectural changes during training. Under a global neuron pruning scheme, merely using pruning ratio as a guidance fells short for this task: each pruning ratio can be easily satisfied by multiple variants, each sharing the same number of neurons while differing in architectures (see examples in Fig. 2). As an alternative, we examine the distribution of the number of remaining neurons across all layers per pruned network.

Consider two sub-networks $\mathcal{N}_1$ and $\mathcal{N}_2$ under the same prune ratio containing the same number of remaining neurons. Let $n_{(1,l)}$ and $n_{(2,l)}$ be the number of neurons of l_{th} layer in nets $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively, then set $\{n_{(1,1)}, n_{(1,2)}, \cdots, n_{(1,L)}\}$ describes the structure of the sub-network $\mathcal{N}_1$, and similarly for $\mathcal{N}_2$. For the l_{th} layer, we define the normalized difference between $\mathcal{N}_1$ and $\mathcal{N}_2$ as

$$d_l(\mathcal{N}_1, \mathcal{N}_2) = \frac{|n_{(1,l)} - n_{(2,l)}|}{n_{(1,l)} + n_{(2,l)}}, \tag{4}$$

yielding a range from zero to one. The lower the distance, the closer the layer structure is. On top of this we can now construct a pruning stability indicator Ψ combining the similarity for all the layers in the network:

$$\Psi(\mathcal{N}_1, \mathcal{N}_2) = 1 - \frac{1}{L}\sum_{l=1}^{L} d_l(\mathcal{N}_1, \mathcal{N}_2), \tag{5}$$

where Ψ ranges from 0 to 1, with a lower value indicates high variations between the two sub-networks, and a high value indicates stability in the resulting network structure.

Given a pruning stability indicator, the algorithm to decide when to prune is described in Algorithm 2. We first calculate the neurons' importance scores according to the pruning criterion at the end of each epoch t. Get the top k neurons by ranking the importance scores and the resultant sub-network structure $\mathcal{N}_t = \{n_{(t,1)}, n_{(t,2)}, \cdots, n_{(t,L)}\}$ where $\sum_{l=1}^{L} n_{(t,l)} = k$ and $n_{t,l}$ is the number of neurons in the l_{th} layer. Then calculate the sub-network structure similarities between $\mathcal{N}_t$ and $\mathcal{N}_{t-j}$ for $1 \le j \le r$ where r is the range of past epochs that we want to have a structure comparison. We use the averaged structure similarity to reflect the structure stability, namely:

$$\mathrm{EPI}_t = \frac{1}{r}\sum_{j=1}^{r} \Psi(\mathcal{N}_t, \mathcal{N}_{t-j}). \tag{6}$$

This structure stability score is constantly increasing during training. When it reaches a certain threshold τ, we can safely say that the resultant sub-network is reliable to achieve a good performance and we can start the pruning.

Algorithm 2 Pruning-aware-Training (PaT)

Input: Network with random initialized weights $\mathbf{W}_{\mathcal{F},0}$, stability threshold τ, pruning ratio α, total epochs T as in original recipe
Output: Pruned structure $\mathcal{R}$; trained weights $\mathbf{W}_{\mathcal{R},T}$

enforce epoch status $\in$ {dense, prune, sparse}
epoch status $\leftarrow$ dense
for epoch $t = 0, 1, \dots, T$ **do**
  if epoch status **is** dense **then**
    Train $\mathbf{W}_{\mathcal{F},t}$ by gradient descent
    Get importance score averaged over the epoch
    Get $(1-\alpha)|\mathcal{F}|$ top ranked neurons to form $\mathcal{N}_t$
    Get EPI_t with Eq. (6)
    if $(\mathrm{EPI}_t \ge \tau)$ and $(\mathrm{EPI}_t \ge \mathrm{EPI}_{t-j})_{1 \le j \le 5}$ **then**
      epoch status $\leftarrow$ prune
    end if
  else if epoch status **is** prune **then**
    Prune $\alpha|\mathcal{F}|$ neurons with Algorithm 1
    Get $\mathcal{P}$, update $\mathcal{R}$
    epoch status $\leftarrow$ sparse
  else
    Train $\mathbf{W}_{\mathcal{R},t}$ by gradient descent
  end if
end for
Return $\mathcal{R}$, $\mathbf{W}_{\mathcal{R},T}$

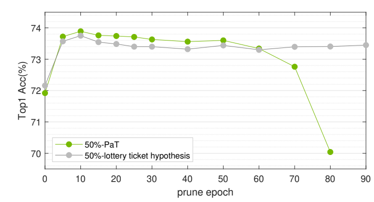


Figure 3. Accuracy as a function of the pruning epoch for prune aware training (PaT) in green, and the lottery ticket in gray for a ResNet50 on ImageNet. The dense version achieves 77.32% accuracy. PaT yields better performance if the pruning starts after a few training epochs. However, if pruning occurs too late, the accuracy for PaT drops significantly.

4. Experiments

We next experiment with varying architectures and pruning methods to showcase the strength of our proposed method for the classification task. In the appendix, we also demonstrate applicability to the task of object detection.

Experimental Settings. We prune ResNet34, ResNet50 and MobileNetV1 neural network architectures on the ImageNet ILSVRC 2012 dataset [34] (1.3M images, 1000 classes). Unless otherwise stated each pruning uses one single node with 8 NVIDIA Tesla V100 GPUs. All experiments share the original training pipeline following PyTorch mixed-precision training under NVIDIA's recipe [32] with 90 epochs in total. The learning rate is warmed up linearly in the first 8 epochs, then follows a cosine decay over the entire training. We use PyTorch Distributed Data Parallel training and for each GPU with an individual batch size at 128. Our unpruned models achieve 77.32% top-1 accuracy with ResNet50, 74.36% with ResNet34 and 72.93% with MobileNetV1.

4.1. Understanding Early Pruning Epochs

We start with understanding in depth the variations in final accuracy as a function of the starting pruning epoch. To do so, we analyze the accuracy changes by varying the starting pruning epoch, and continue training to the final epoch and check the associated accuracies.

Pruning at different epochs. Fig. 3, in green, shows the top-1 accuracy obtained by pruning 50% of the neurons on a ResNet50 using gradient-based criterion at various epochs during the 90-epoch training cycle. As shown, the accuracy drop for late pruning is significant as there is not enough time left for recovering. We also observe that, compared to pruning at initialization (at epoch 0), pruning after a few epochs consistently yields better performance. However, for all these experiments there is always a certain accuracy drop compared to the unpruned upper bound (77.32%).

Lottery ticket hypothesis for structural pruning. To better understand the role of early training with a dense model rather than a pruned model, we evaluate the idea of lottery-ticket hypothesis for structural pruning. We follow [9] and train from scratch a sub-network obtained by pruning using the original initialization. All pruning masks are collected from the previous experiment (Fig. 3). Results are shown in the same plot with a gray line. Note that, due to iterative nature of the pruning algorithm, for pruning at 0 we use the mask when the epoch is finished. When it is applied at the initialization as a lottery ticket, the final accuracy is slightly different. From these results, we can conclude that, in the structural pruning case, the lottery ticket hypothesis may not hold. Pruning the network during training performs better than training a winning ticket in isolation from scratch.

Varying pruning ratio and architecture. We now take a closer look at the accuracy drop incurred when pruning occurs during the early stage of training (first 30 epochs from Fig. 3). Fig. 4 shows these results for different architectures using magnitude-based and gradient-based pruning respectively. As we can see, for magnitude-based pruning, there is a significant drop in accuracy if the pruning occurs too early, especially for large prune ratios. This effect is particularly clear if, for instance, we prune 50% neurons of a ResNet50 at epoch 0 which makes the network not trainable. This is expected as the pruning ratio is large and the weights have not been updated at all. Therefore, it is not possible to estimate the importance of each neuron correctly. For gradient-based pruning, the accuracy drop varies depending on the architecture. In this case, pruning at initialization has less impact compared to magnitude-based pruning. For instance, pruning a ResNet34 or a MobileNetV1 leads to minimal drop in accuracy. For ResNet50, however, the accuracy drop increases as the pruning ratio increases. Thus, late pruning would be preferred to maximize performance. Let's see how we can find the optimal pruning epoch during a single training session.

4.2. EPI-guided Pruning

Given the previous results, we now demonstrate the ability of our approach to determine the optimal pruning epoch. Thus, in this experiment we compare our policy to a heuristic and a random policies. For heuristic policy, we consider setting the pruning epoch to 0 which is equivalent to pruning at initialization [8, 24]. For the random policy, we select randomly a pruning epoch during the early stage of training, *i.e.* in the range $[0, 30]$. We repeat the random policy experiment 100 times and report the mean of the results.

Selecting the EPI threshold (τ). Our method introduces a hyperparameter τ such that when EPI (Eq. 6) reaches it we can start pruning. We find that a universal value can be used for all architectures and all pruning ratios, however, it is sensitive to the pruning algorithm. To this end, we perform a sensitivity analysis on ResNet50 over pruning ratios of 10%–50% with increments of 10% and use a grid search to set the value that yields the best pruning result. As a result, we find $\tau = 0.983$ for magnitude-based pruning and $\tau = 0.944$ for gradient-based pruning to be the best. We tuned this value for ResNet50, however, we will show its generalizability by performing tests on ResNet34, MobileNetV1 in the main text and SSD in the Appendix.

	Network(s)	Starting epoch for pruning			
		Random	Init. [8, 24]	Pre-defined [11]	EPI (ours)
Magnitude	ResNet50	0.940	3.604	0.115	**0.091**
	ResNet34	0.267	0.838	0.353	**0.169**
	MobileNet-v1	6.285	–	0.135	**0.135**
	Overall	2.497	2.221	0.201	**0.132**
Gradient	ResNet50	0.992	2.738	0.267	**0.092**
	ResNet34	0.153	**0.122**	0.221	0.195
	MobileNet-v1	0.132	0.186	**0.110**	0.178
	Overall	0.426	1.015	0.199	**0.155**

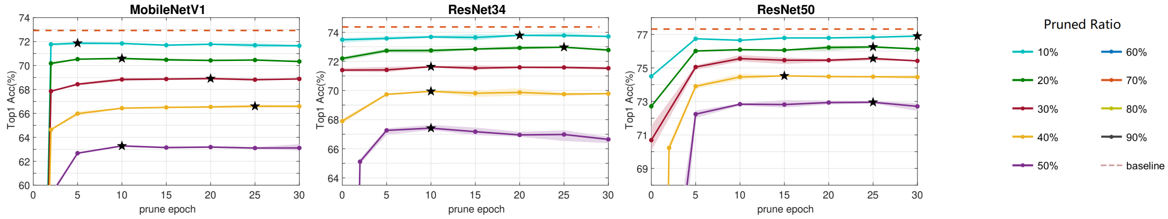
Table 1. Absolute top-1 accuracy change relative to the results per oracle grid search for optimal pruning starting point. Lower is better. Init. refers to pruning at initialization using a structural version of FORCE [8], also equivalent to the structural and iterative SNIP [24]. Pre-define refers to heuristically pre-define a pruning start epoch 30, which we find empirically the last epoch not leading to a significant accuracy drop. This is aligned with [10] suggesting waiting for several training iterations before pruning.

Policy comparison. Tab. 1 shows the results of experiments for importance and magnitude based pruning under the guidance of our proposed EPI and the universal EPI threshold τ. We compare the results with random policy and heuristic policy of pruning at initialization. We report the average accuracy drop compared to best accuracy achieved via grid search for each network. The values for ResNet50 gradient-based pruning is averaged over prune ratios 10%–90%; all the rests are averaged over prune ratios 10%–50%. We also report the overall average accuracy drop over three networks when using different policies to guide the pruning in row "overall". As shown, our policy clearly yields a significantly higher performance compared to random pruning. In the case of gradient-based pruning, our approach performs on par compared to heuristics on ResNet34 and MobileNetV1. Overall our approach performs better with less top-1 accuracy change. For magnitude-based pruning, our approach yields significantly better results compared to heuristics. The optimal pruning epoch varies for different architectures and different prune ratios, see the appendix for the sensitivity analysis to the stability threshold.

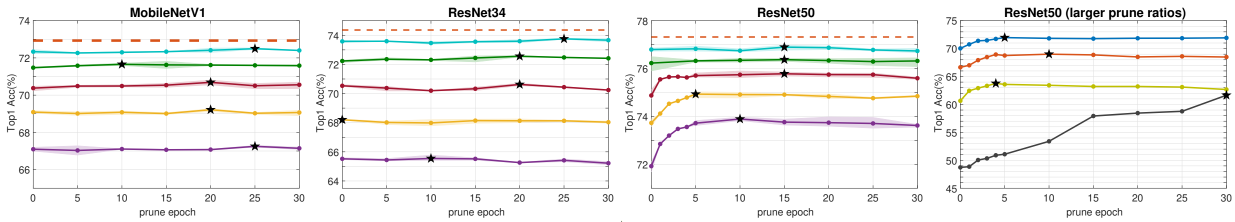
4.3. Training Speedup

We also calculate the actual training speedup when using our proposed EPI (Eq. 6) policy and the heuristic ones on ResNet50 gradient-based pruning. Fig. 5 shows the policies comparison result.

Comparing to pruning at initialization. When compared with pruning at initialization (heuristic pruning at epoch 0), we achieve larger speed-up although we start prune later. That happens because of structural pruning, where pruning at different epochs might result in different structures: faster



(a) Magnitude-based pruning



(b) Gradient-based pruning

Figure 4. The final ImageNet Top-1 accuracy of the pruned network when pruning occurs at different epochs during the early stage of training. We observe pruning at initialization tends to result in untrainable network with magnitude-based pruning method. For gradient-based method, we observe a higher degradation occur when more filters are pruned, and show pruning ratios up to 90% on ResNet50. [Pruned Ratio] denotes the percentage of neurons removed. Results on MobileNetV3 are shown in the appendix.

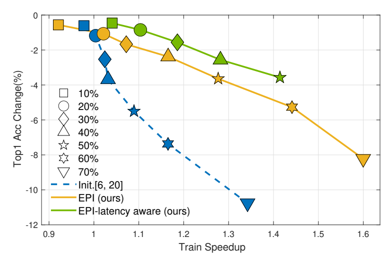


Figure 5. Actual training speed-ups on ResNet50 with different prune ratios using different pruning policies. Actual speed measured on an NVIDIA TITAN V GPU at batch size 64. Top-right corner is preferred.

or slower. It turns out that pruning at epoch 0 is very inefficient, as most neurons pruned are from deeper layers resulting in slow models; therefore, training speed-up is not huge and training such a model is more expensive than the one resulted from pruning at a later epoch. In the meantime, pruning at zero leads to larger accuracy drop especially with large prune ratios.

Latency-aware pruning. We further apply latency-aware pruning using our policy, which aims to reduce the latency of the model and not only the number of parameters. For that, we penalize the neuron group's saliency with the latency reduction resulting from pruning them. Those neurons requiring larger compute will have lower importance and, therefore, more likely to be pruned. As shown in Fig. 5, using EPI-latency aware pruning yields models that are more GPU-friendly and faster.

Training cost comparison. Training a dense ResNet50 on ImageNet with 8 NVIDIA Tesla V100 GPUs takes around

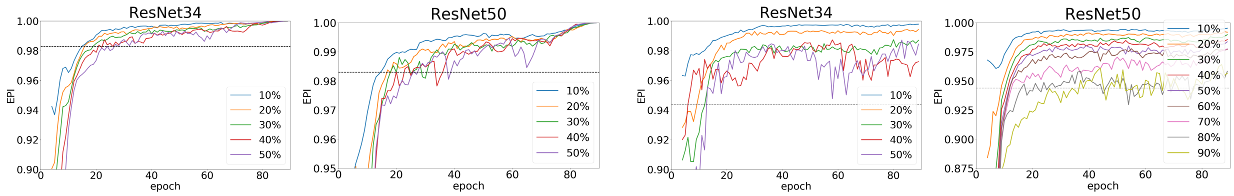
	Method	Top-1 acc. ↑	FLOPs (G)↓	Starting epoch ↓	Total train FLOPs reduc.(%) ↑
ResNet34	GPWP [2]	**73.64**	3.087	40	13.8
	PaT **(ours)**	73.50	**2.911**	**11**	**25.6**
ResNet50	PT [27]	74.73	2.303	-	30.0
	LFPC [17]	74.18	**1.612**	35	37.2
	PaT **(ours)**	**74.85**	1.695	**13**	**50.3**

Table 2. Comparison with state-of-the-art in-training pruning methods. For a fair comparison we report here 10% filter pruning for ResNet34 and 40% filter pruning for ResNet50 as literature.

9 hours and costs around \$220 on AWS. Considering a 50% pruning ratio, the training cost of our method is \$154 as we achieve up to 30% training time reduction. In contrast, the total training cost for train-prune-finetune methods is around \$364 as they need 90 additional retraining epochs. Thus, a 2.4× training cost reduction.

4.4. Comparisons with the State-of-the-art

We also compare our method to prior arts on ImageNet dataset and present results in Tab. 2. Compared to GPWP [2], our method yields lower accuracy but a higher reduction in FLOPs. Compared to PruneTrain [27], we achieve higher accuracy and a significantly larger training FLOPs saving. LFPC [17] yields 0.67% larger accuracy loss. Even with less FLOPs remaining in pruned model, LFPC requires 13% more total training FLOPs as it needs more epochs for dense train. Overall, we prune much earlier than other techniques, and, therefore, reduce more training time. Moreover, while other techniques use heuristics to define the pruning epoch, we propose an automatic metric that scales with a universal threshold to automatically determine an early point. This helps create a new benchmark protocol by evaluating prior methods under the same setting.



(a) magnitude-based EPI (b) gradient-based EPI

Figure 6. Structure stability analysis for ResNet architectures for (a) magnitude-based and (b) gradient-based pruning. Dashed line shows the EPI threshold selected for each network under the pruning criterion. Results for MobilnetV1 can be found in the supplemental material.

4.5. Ablation Studies

Stability analysis. The goal of this experiment is to demonstrate that during the early stage of training, the dominate sub-network architecture varies significantly and slowly converges to the final architecture as the training progresses. To this end, we train different architectures to convergence and, in the process, we compute their EPI scores for sub-networks under different pruning ratios. Fig. 6 shows the results for this experiment for magnitude-based and gradient-based criterion respectively. As shown, the EPI score decreases as the pruning ratio increases. We also observe that, independently of the pruning method used, the stability grows rapidly in the early stage of training and then continues increasing steadily for later stages. These results are consistent with those presented in [11] showing a significant change in the network architecture at the beginning but not towards the end of training.

Performance with similar structures. We now provide empirical support to the assumption that sub-networks with similar structures will likely perform similarly. That is, the performance relies on the structure rather than the neurons being selected during pruning. We use the ResNet50 and obtain the pruning mask with 50% neurons off at epoch 10 using gradient-based ranking. Next, we obtain 10 variations of this mask by selecting different neurons in each layer while maintaining the number of neurons per layer. As a result, we get different neuron masks but the same sub-network structure. We train the initially pruned network to convergence and obtain a top1 accuracy of 73.98%. Finally, we evaluate the performance of training to completion the checkpoint pruned using the 10 mask variations yielding an average top1 accuracy difference of only $0.36\% \pm 0.13\%$. We also randomly generate 5 variations of masks that lead to different sub-network structures, which have an around 0.8 similarity score to the structure resulted from the original mask. We get an average performance difference of $0.53\% \pm 0.39\%$. Note that, for a fair comparison, we use the same checkpoint for prune to minimize randomness due to different initialization. From these results, we can conclude networks with the same structure perform similarly. With different structures, the performance varies more.

Compared to pruning pre-trained models. In many practical scenarios, people train a model (or use a pre-trained backbone) to solve the task at hand. However, there are computation or latency constraints at deployment. People might want to train a smaller model from scratch or prune and finetune the existing model. Both scenarios would benefit from EPI, as we show next.

prune ratio	post trained pruning	EPI, equal epochs	EPI, equal training time
10%	77.17%	**77.69%**	**77.74%**
20%	76.78%	**76.82%**	**76.83%**
30%	76.18%	**76.23%**	**76.55%**

Table 3. Comparison between pruning during training with EPI and pruning a pre-trained model.

We compare pruning during training versus pruning pre-trained models in Tab. 3. For pruning a pre-trained model (loading Pytorch weights from ResNet50 trained for 90 epochs, with acc. 77.32%), we do pruning in the same setting and then finetune it for 90 epochs (marked as *post trained pruning*). For a fair comparison when we apply EPI on a model trained from scratch we increase the total number of epochs to $90+90 = 180$ (column *EPI, equal epochs*). We clearly outperform pruning pre-trained weights, while saving compute costs. We get pruned models earlier, making the training even faster. Models from EPI can be finetuned for the same amount of clock time as *post trained pruning*, shown as column *EPI, equal training time*. This improves results even more as models get trained for more epochs. Again, applying EPI speeds up training right after pruning is complete, while pruning on pre-trained benefits pruning only after the training is finished.

5. Conclusions

We have introduced an approach to automatically determine when pruning can be performed during training without affecting the final accuracy and with the additional constraint of doing so as early as possible. To this end, we have proposed a policy based on Early Pruning Indicator (EPI), a metric to measure the stability of the sub-network structure. Our experiments on multiple pruning algorithms and pruning ratios have demonstrated the benefits of our method to reduce the accuracy drop when pruning a network and observe a significant reduction in training time.

Limitations. Our experiments are mainly focused on image classification and object detection (in supplementary material). We leave for future work checking the generalizability of our policy to other learning tasks.

References

[1] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018. 2

[2] S. A. Aketi, S. Roy, A. Raghunathan, and K. Roy. Gradual channel pruning while training using feature relevance scores for convolutional neural networks. *IEEE Access*, 8:171924–171932, 2020. 3, 7

[3] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016. 1, 2, 3, 4

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 2

[5] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020. 1

[6] Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020. 3

[7] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, P Vajda, M Uyttendaele, and Niraj K Jha. ChamNet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019. 1

[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. *arXiv preprint arXiv:2006.09081*, 2020. 2, 3, 4, 6

[9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 2, 3, 5

[10] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, 2020. 2, 3, 6

[11] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. The early phase of neural network training. In *International Conference on Learning Representations*, 2020. 2, 4, 6, 8

[12] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. In *International Conference on Machine Learning*, 2020. 2

[13] Susan Gao, Xin Liu, Lung-Sheng Chien, William Zhang, and Jose M Alvarez. Vacl: Variance-aware cross-layer regularization for pruning deep residual networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 3

[14] Yue-Seng Goh and Eng-Chong Tan. Pruning neural networks during training by backpropagation. In *Proceedings of TENCON'94-1994 IEEE Region 10's 9th Annual International Conference on:'Frontiers of Computer Technology'*, pages 805–808. IEEE, 1994. 3

[15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015. 1, 4

[17] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020. 2, 3, 7

[18] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019. 2

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1

[20] A. Howard, Menglong Zhu, Bo Chen, D. Kalenichenko, W. Wang, Tobias Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017. 4

[21] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[22] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. 3

[23] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. volume 97 of *Proceedings of Machine Learning Research*, pages 3519–3529, 09–15 Jun 2019. 3

[24] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019. 2, 3, 4, 6

[25] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. 1, 2, 3

[26] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017. 1, 2

[27] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019. 2, 3, 7

[28] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 1

[29] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 1, 2, 3, 4

[30] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 1, 2, 3, 4

[31] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3573–3582, 2019. 1

[32] Nvidia. Convolutional networks for image classification in pytorch. `https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets#convolutional-networks-for-image-classification-in-pytorch`, 2020. 5

[33] Oyebade Oyedotun, Djamila Aouada, and Bjorn Ottersten. Structured compression of deep neural networks with debiased elastic group lasso. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2277–2286, 2020. 2, 3

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015. 5

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 1

[36] Charles Spearman. The proof and measurement of association between two things. 1961. 3

[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 4

[38] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1

[39] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 1

[40] Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020. 3

[41] Arash Vahdat, Arun Mallya, Ming-Yu Liu, and Jan Kautz. Unas: Differentiable architecture search meets reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11266–11275, 2020. 1

[42] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. 2, 3

[43] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2078–2087, 2020. 1

[44] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 1

[45] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via DeepInversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020. 1

[46] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016. 1