The Simpler The Better: An Entropy-Based Importance Metric to Reduce Neural Networks' Depth

Victor Quétu(✉), Zhu Liao, and Enzo Tartaglione

LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France
{victor.quetu,zhu.liao,enzo.tartaglione}@telecom-paris.fr

Abstract. While deep neural networks are highly effective at solving complex tasks, large pre-trained models are commonly employed even to solve consistently simpler downstream tasks, which do not necessarily require a large model's complexity. Motivated by the awareness of the ever-growing AI environmental impact, we propose an efficiency strategy that leverages prior knowledge transferred by large models. Simple but effective, we propose a method relying on an **E**ntropy-b**AS**ed **I**mportance m**E**t**R**ic (**EASIER**) to reduce the depth of over-parametrized deep neural networks, which alleviates their computational burden. We assess the effectiveness of our method on traditional image classification setups. Our code is available at https://github.com/VGCQ/EASIER.

Keywords: Compression · Efficiency · Deep Learning

1 Introduction

Deep Neural Networks (DNNs) have drastically changed the field of computer vision. They have been crucial in obtaining state-of-the-art results in several important computer vision domains, such as semantic segmentation [8], classification [26], and object detection [47]. Beyond traditional computer vision tasks, DNNs have also impacted other fields by exhibiting unbridled potentials in natural language processing [45], and multi-modal tasks [41]. DNNs' use is growing significantly in our lives and appears to be perennial.

Despite DNNs have demonstrated scalability in terms of model and dataset size [21], they hinder high computational demands. Indeed, neoteric architectures are made up of millions, or even billions, of parameters, resulting in billions, or even trillions, of FLoating-point OPerations (FLOPs) for a single inference [17]. Hence, these large models require enormous resources both in terms of pure hardware capacity and energy consumption, for training and deployment, which raises issues for real-time and on-device applications and also has an environmental impact. For instance, GPT-3 [6], made of 175B parameters, emits around

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-70365-2_6.

A. Bifet et al. (Eds.): ECML PKDD 2024, LNAI 14946, pp. 92–108, 2024.
https://doi.org/10.1007/978-3-031-70365-2_6

200tCO_2eq for its training and its operational carbon footprint reached around 550tCO_2eq [14].

The development of compression techniques, which constitute an essential means of remedying the resource-hungry nature of DNNs, has marked the research landscape over the past decade. It is well-known that the complexity of the model is intrinsically linked to the generalizability of DNNs [21], and since pre-trained architectures that can be used in downstream tasks tend to be over-parameterized, compression with no (or only slight) performance degradation is in principle possible [43]. To design a more efficient architecture, a set of methods has been proposed, ranging from parameter pruning [18] to the reduction of numerical precision [37]. Nonetheless, few approaches are capable of lessening the number of layers in a DNN. Indeed, removing single parameters or whole filters offers very few if any, practical benefits when it comes to using the model on recent computing resources, such as GPU. Thanks to the intrinsic parallel computation nature of GPUs or TPUs, the limitation on layer size, whether larger or smaller, comes mainly from memory caching and core availability.

In most cases, this parallelization capability avoids the need to reduce layer size, suggesting that another approach needs to be explored to address this problem. Indeed, reducing the critical path that computations must traverse [2] would help to relieve the DNN's computation demand, which can be achieved by strategically removing layers. Despite that existing approaches, like knowledge distillation [22], implicitly tackle this issue, the absence of performance degradation cannot be guaranteed, since a shallow target model is imposed. This motivates the exploration of designing a method for neural networks' depth reduction while preserving optimal performance.

In this work, we present our method EASIER, which iteratively tries to reduce the depth of deep neural networks. More precisely, EASIER identifies the average state of a given rectifier-activated neuron for the trained task. Given the definition of rectifier activation functions, EASIER can find the probability that this neuron uses one of the two regions, and hence can calculate an entropy-based metric per layer. Such a metric is then used to drive the linearization of layers toward neural network depth reduction. We summarize, here below, our key messages and contributions.

- We highlight how we can potentially reduce the depth of a neural network with a marginal impact on the performance by characterizing layer degeneration (Sect. 3.1).
- We propose EASIER, a method relying on an entropy-based importance metric that pinpoints rectifier-activated layers that can be linearized (Sect. 3.3) (Fig. 1).
- We test EASIER across multiple architectures and datasets for traditional image classification setups (Sect. 4), demonstrating that layer withdrawal can be achieved with little or no performance loss when over-parameterized networks are employed. Notably, we show the potential savings in terms of FLOPs and inference time on six different hardwares, highlighting the benefits of our method (Sect. 4.3).

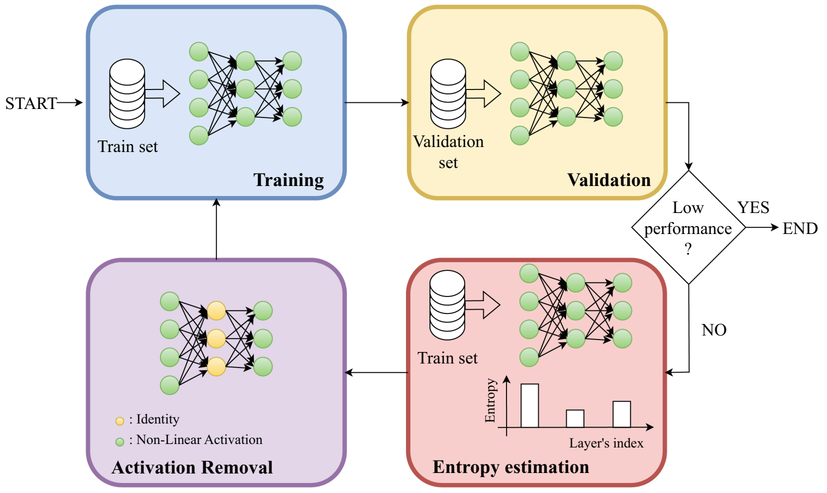

Fig. 1. Overview of EASIER. We iteratively train, evaluate, and estimate the entropy on the training set and linearize the lowest-entropy layer of the neural network, until the performance drops.

2 Related Works

Neural Architecture Search. Popular deep neural network architectures have mostly been designed by hand, among which we can cite VGG [40], ResNet [19], MobileNet [23] or Swin transformer [31]. Despite leading to remarkable performance on a variety of tasks, the design of novel architectures is time-consuming and can be prone to errors. Neural Architecture Search (NAS) was the answer to both these problems. Divided into subgroups such as evolutionary methods [34], methods based on reinforcement learning [49], and differentiable methods [30], NAS is finding the contemporary top-performing architectures [3]. While the firsts are based on efficient heuristic search methods based on evolution to capture global solutions of complex optimization problems [38], the second relies on goal-oriented optimization methods driven by an impact response or signal [1]. Differentiable methods learn architectural paths that enable the removal of entire layers and sometimes add width to the previous ones to balance [46]. By disentangling training and searching to reduce the cost, a popular approach proposed a large once-for-all network [7] supporting diverse architectural designs. The idea was to select a sub-network within the aforementioned model without the need for additional training.

Nonetheless, despite reducing the model size across diverse dimensions, like depth, width, kernel size, and resolution, NAS approaches, including also this work, generally need expensive computational resources to span several search space dimensions and train a super-network from scratch. In this paper, our sole focus lies on depth as the exclusive search dimension, by leveraging a pre-trained model, making easier convergence and reducing the overall training time.

Neural Network Pruning. Neural network pruning, whose goal is to shrink a large network to a smaller one while maintaining performance by removing irrelevant weights, filters, or other structures from neural networks, has gained significant attention in neoteric works since it allows a possible model performance enhancement and an over-fitting reduction. On the one hand, *structured* pruning focuses on removing entire neurons, filters, or channels [20,44]. On the other hand, *unstructured* pruning algorithms discard weights without explicitly taking the neural network's structure into account [18,43]. The main categories of unstructured pruning methods are magnitude-based pruning [18,32,48] and gradient-based pruning [28,43]. While the first eponymous approach takes the weights' magnitude as an importance score to prune parameters, the latter uses the gradient magnitude (or its higher-order derivatives) to rank them. The effectiveness of these techniques was compared by [4] and, in general, magnitude-based methods are more accurate than gradient-based. Moreover, they are a good trade-off between complexity and competitiveness. Indeed, [15] exposed that simple magnitude pruning approaches reach similar or better results than complex methods. From a computational perspective, in a general-purpose hardware configuration, larger benefits in terms of both memory and computation are produced by structured pruning compared to unstructured pruning, even though the reached sparsity can be significantly lower [5].

However, a recent work [29] proposed an unstructured Entropy-Guided Pruning (EGP) algorithm, that succeeds in reducing the depth of deep neural networks by prioritizing pruning connections in low-entropy layers, leading to their entire removal while preserving performance. Our method differs from the latter since EASIER considers a third state to calculate the entropy (Sect. 3) and unlike EGP, our method does not involve pruning. Although effective, EGP only allows a small number of layers to be removed. Indeed, after the removal of multiple layers, the accuracy drops dramatically. This will be verified by comparing this method with EASIER, in Sect. 4.

Activation Withdrawal. Private inference has led to an upsurge in works on removing non-linear activations. Indeed, a high latency penalty is incurred when computing on encrypted data, which is mainly due to non-linear activations such as ReLU. Methods such as DeepReduce [24] and SNL [9] have been developed to reduce private inference latency. While DeepReduce includes both optimizations for ReLU dropping and knowledge distillation training to maximize the performance, the latter proposes a gradient-based algorithm that selectively linearizes ReLUs while maintaining prediction accuracy. However, although SNL significantly reduces the number of ReLU units in the neural network, it never removes activation from an entire layer, but only from units such as pixels or channels. In contrast, our method focuses on removing activation functions at a layer level, in order to reduce the depth of deep neural networks. Moreover, DeepReduce [24] is based on a criticality metric requiring five optimized networks per optimization iteration, resulting in the exploration of $5 \times (D - 1)$ network architectures, for a network with D stages, which is not very efficient at training time. On the other hand, our method does not require leveraging the knowledge of a teacher model to boost performance, saving computation at training time. Although left for future work, we believe our work can also be effective in accelerating private inference.

In traditional classification setups, [13] introduced Layer Folding, a technique that determines whether non-linear activations can be withdrawn, enabling the folding of adjacent linear layers into one. More specifically, PReLU activations with a trainable slope, replace ReLU-activated layers. The almost linear PReLUs are eliminated post-training, enabling the layer to be folded with its successive one. Furthermore, a comparable channel-wise method enabling a notable reduction in non-linear units in the neural network while preserving performance was put forward by [2]. While the latter does not aim at reducing neural network depth, Layer Folding was originally proposed only for ReLU-activated networks. Designed for any rectifier, we will compare our method EASIER with Layer Folding and demonstrate its effectiveness in Sect. 4.

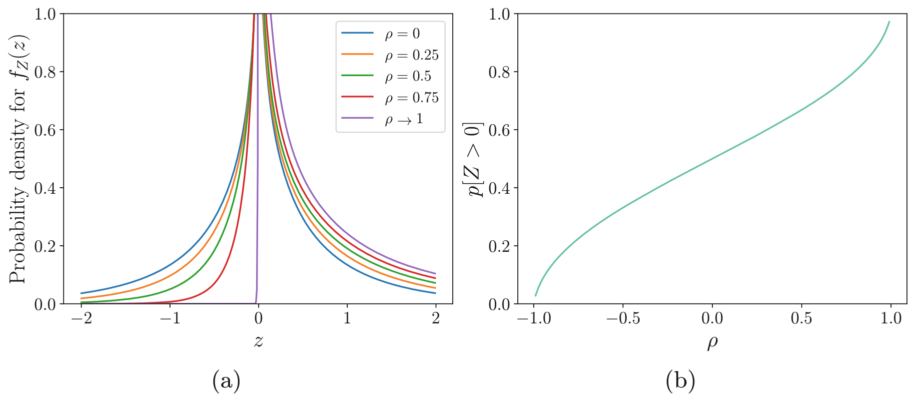

Fig. 2. Distribution of the product between $X \sim \mathcal{N}(0,1)$ and $W \sim \mathcal{N}(0,1)$ for different values of ρ (a), and $p[Z > 0]$ for different ρ (b).

3 Method

In this section, we first highlight how we can potentially reduce the depth of a neural network with a marginal impact on performance. Based on this observation, we then derive an entropy formulation for rectifier activations, which will be at the heart of our EASIER method.

3.1 How Layers Can Degenerate

Let us define the input $\boldsymbol{x}$ for a given neuron is a sequence of random variables $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$. Similarly, we can assume the N parameters populating such neuron, for a large N limit, follow as well a Gaussian distribution, and we model it as $W \sim \mathcal{N}(\mu_W, \sigma_W^2)$. Under the assumption of $\mu_X = \mu_W = 0$ (for narration purposes, it is possible to derive a more general result according to [12]), we can obtain the distribution for the pre-activation z (resulting from the product of the weights and the input, modeled through the random variable Z), according to the result obtained by [11,12,39], follows the probability density function

$$f_Z(z) = \frac{1}{\pi \sigma_X \sigma_W \sqrt{1-\rho^2}} \exp\left[\frac{\rho z}{\sigma_X \sigma_W (1-\rho^2)}\right] K_0 \left[\frac{|z|}{\sigma_X \sigma_W (1-\rho^2)}\right], \quad (1)$$

where K_n is the n-th order modified Bessel function of the second kind and ρ is the correlation coefficient between X and W. A visual representation of its distribution is pictured in Fig. 2a. We can clearly observe the large impact of ρ, steering how the values will effectively be distributed. Now, let us assume the activation function of such a neuron is a rectifier function, and we are interested in observing what is the probability of the post-activation output being in the linear region: we are interested in measuring $p[Z > 0] = 1 - F_Z(0)$, where $F_Z(x) = p[Z < x]$ is the cumulative distribution function (CDF) for the density $f_Z(z)$. A visual representation of how these values are distributed for different values of ρ is depicted in Fig. 2b.

The behavior of neurons, particularly when employing rectifiers like ReLU, is tightly linked to the learning process, and W becomes more and more (anti-) correlated with X. At $\rho \to 1$, neurons operate linearly, leading to a layer's degeneration (as the current layer becomes a linear combination with the next layer). Conversely, at $\rho \to -1$, neurons become effectively "OFF", leading to insignificance in their contribution. In both cases, there's a *layer degeneration* that we aim to detect to reduce the neural network's depth with a marginal impact on the performance. In the next section, we will draft a metric to estimate how close a layer is to degenerating.

3.2 Entropy for Rectifier Activations

To monitor the output $y_{l,i}^{\boldsymbol{x}}$ of the i-th neuron from a given input $\boldsymbol{x}$ of the dataset $\mathcal{D}$, we define ψ_l as the rectifier of the l-th layer, populated by N_L neurons. Hence, by assuming that $z_{l,i}^{\boldsymbol{x}}$ is the output of the i-th neuron inside the l-th layer, we obtain:

$$y_{l,i}^{\boldsymbol{x}} = \psi_l(z_{l,i}^{\boldsymbol{x}}), \quad (2)$$

Three possible "states" for the neuron can be identified from (2):

$$s_{l,i}^{\boldsymbol{x}} = \begin{cases} +1 \text{ if } y_{l,i}^{\boldsymbol{x}} > 0 \\ -1 \text{ if } y_{l,i}^{\boldsymbol{x}} < 0 \\ 0 \quad \text{if } y_{l,i}^{\boldsymbol{x}} = 0. \end{cases} \quad (3)$$

More precisely, for the output of the i-th neuron, by simply applying the sign function to $z_{l,i}^{\boldsymbol{x}}$, we get $s_{l,i}^{\boldsymbol{x}} = \text{sign}(z_{l,i}^{\boldsymbol{x}})$ and can hence easily pinpoint in which of these states we are. Candidly, the neuron is in the *ON state* when $s_{l,i}^{\boldsymbol{x}} = +1$, as this generally corresponds to the linear region, as opposed to the *OFF state* when $s_{l,i}^{\boldsymbol{x}} = -1$ (considering that $\lim_{x \to -\infty} \psi(x) = 0$).[1] Since it could belong to either the ON or OFF state, the third state $s_{l,i}^{\boldsymbol{x}} = 0$ is a special case, which will not be considered in the following derivation.

The probability (in the frequentist sense) of the i-th neuron belonging to either the ON or the OFF state can be calculated from the average over a batch of outputs for this neuron. More precisely, we define the ON state probability as:

$$p(s_{l,i}\!=\!+1) = \begin{cases} \dfrac{1}{S_{l,i}} \displaystyle\sum_{j=1}^{|\mathcal{D}|} s_{l,i}^{\boldsymbol{x}_j} \Theta(s_{l,i}^{\boldsymbol{x}_j}) & \text{if } S_{l,i} \neq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where

$$S_{l,i} = \sum_{j=1}^{|\mathcal{D}|} s_{l,i}^{\boldsymbol{x}_j} \operatorname{sign}(s_{l,i}^{\boldsymbol{x}_j}) \tag{5}$$

is the frequency of the ON and the OFF states encountered, $|\mathcal{D}|$ is the number of input samples, and Θ is the Heaviside function.[2] As explained above, the third state is excluded from this count, as it can be associated with either the ON or OFF state. We can therefore infer that since we are just concerned with the ON or OFF states, when $S_{l,i} \neq 0$, $p(s_{l,i}\!=\!-1) = 1 - p(s_{l,i}\!=\!+1)$. We define as an estimator for *neuron's degeneration* the entropy of the i-th neuron in the l-th layer, calculated as:

$$\mathcal{H}_{l,i} = -\sum_{s_{l,i}=\pm 1} p(s_{l,i}) \log_2 \left[p(s_{l,i})\right] \tag{6}$$

Given the definition in (6), $\mathcal{H}_{l,i} = 0$ can be verified in two cases:

– $s_{l,i} = -1\ \forall j$. In this case, $z_{l,i} \leq 0\ \forall j$. The output of the i-th neuron is always 0 when for example employing a ReLU.
– $s_{l,i} = +1\ \forall j$. In this case, $z_{l,i} \geq 0\ \forall j$. As it belongs to the linear region, the output of the i-th neuron is equal to its input (or very close as in GeLU). Therefore, since there is no non-linearity between them anymore, this neuron can in principle be absorbed by the following layer.

Please note that the case $z_{l,i} = 0\ \forall j$, can be associated with both cases, as mentioned previously, and is therefore not taken into account in the previous case disjunction.

[1] Few exceptions to this exist, like LeakyReLU. In those occurrences, even though the activation will not converge to zero, we still choose to call it OFF state as, given the same input's magnitude, the magnitude of the output is lower.

[2] Please be aware that additional sum and average over the entire feature map generated per input are required for convolutional layers.

As an estimator for *layer's degeneration* we can employ the average entropy: for the l-th layer counting N_l neuron it is

$$\widehat{\mathcal{H}}_l = \frac{1}{N_l} \sum_i \mathcal{H}_{l,i}. \tag{7}$$

We would like to have $\widehat{\mathcal{H}}_l = 0$ since we target deep neural networks' depth reduction by eliminating layers with almost zero entropy. In the next section, we will present the whole framework that allows us to practically reduce the network's depth based on the layer degeneration estimator.

Algorithm 1. Our proposed method EASIER.

```
1: function EASIER($\boldsymbol{w}^{\text{INIT}}$, $\mathcal{D}$, $\delta$)
2:     $\boldsymbol{w}$ ←Train($\boldsymbol{w}^{\text{init}}$, $\mathcal{D}_{\text{train}}$)
3:     dense_acc ←Evaluate($\boldsymbol{w}$, $\mathcal{D}_{\text{val}}$)
4:     current_acc ← dense_acc
5:     while (dense_acc - current_acc) > $\delta$ do
6:         $\widehat{\mathcal{H}} = [\widehat{\mathcal{H}}_1, \widehat{\mathcal{H}}_2, ..., \widehat{\mathcal{H}}_L]$        ▷ Entropy calculation on $\mathcal{D}_{\text{train}}$
7:         $l$ ← argmin($\widehat{\mathcal{H}}$)        ▷ Finding the lowest-entropy layer
8:         $\psi_l$ = Identity()        ▷ Replacement of the rectifier with an Identity
9:         $\boldsymbol{w}$ ← Train($\boldsymbol{w}$, $\mathcal{D}_{\text{train}}$)        ▷ Finetune
10:        current_acc ← Evaluate($\boldsymbol{w}$, $\mathcal{D}_{\text{val}}$)
11:    end while
12:    return $\boldsymbol{w}$
13: end function
```

3.3 EASIER

Depicted in Algorithm 1, we present here our method to remove the lowest-entropy layers. Indeed, the lowest-entropy layer is the one likely to make the least use of the different regions, or states, of the rectifier. Therefore, the need for a rectifier is reduced: the rectifier can be linearized entirely. In this regard, we first train the neural network, represented by its weights at initialization $\boldsymbol{w}^{\text{init}}$, on the training set $\mathcal{D}_{\text{train}}$ (line 2) and evaluate it on the validation set $\mathcal{D}_{\text{val}}$ (line 3). As defined in (7), we then calculate the entropy $\widehat{\mathcal{H}}$ on the training set $\mathcal{D}_{\text{train}}$ for all the L rectifier-activated layers, (therefore, the output layer is excluded) (line 6). We then find the lowest-entropy layer (line 7) and replace its activation with a linear one, i.e., the Identity function (line 8). Evidently, after this step, this layer is not considered anymore. To recover the potential performance loss, the model is then finetuned using the same policy (line 9) and re-evaluated on the validation set $\mathcal{D}_{\text{val}}$ (line 10). The final model is obtained once the performance on the validation set drops below the threshold δ.

4 Experiments

In this section, we empirically evaluate the effectiveness of our proposed approach, across multiple architectures and datasets for traditional image classification setups. We compare our results with EGP [29], an entropy-guided unstructured pruning technique, as well as the Layer Folding method [13].

Table 1. Test performance (top-1) and the number of removed layers (Rem.) for all the considered setups. Dense refers to the original trained model without layer deletion. The best results between LF, EGP, and EASIER are in **bold**.

Dataset	Approach	ResNet-18		Swin-T		MobileNetv2		VGG-16	
		top-1	Rem.	top-1	Rem.	top-1	Rem.	top-1	Rem.
CIFAR-10	Dense	92,47	0/17	91,66	0/12	93,65	0/35	93,50	0/15
	LF	90,65	1/17	85,73	2/12	89,24	9/35	86,46	3/15
	EGP	92,00	3/17	86,04	6/12	92,22	6/35	10,00	1/15
	EASIER	**92,10**	**8/17**	**91,41**	**7/12**	**93,16**	**12/35**	**93,61**	**8/15**
Tiny ImageNet 200	Dense	41,26	0/17	75,78	0/12	46,54	0/35	63,94	0/15
	LF	37,86	4/17	50,54	1/12	25,88	12/35	31,44	6/15
	EGP	39,82	4/17	67,38	3/12	47,52	6/35	—	—
	EASIER	**40,42**	4/17	**68,46**	3/12	**48,80**	**28/35**	**57,60**	**7/15**
PACS	Dense	79,70	0/17	97,30	0/12	95,50	0/35	95,40	0/15
	LF	82,90	3/17	87,70	2/12	79,70	1/35	93,60	3/15
	EGP	81,60	3/17	93,50	4/12	17,70	3/35	—	—
	EASIER	**84,30**	**13/17**	**94,30**	4/12	**94,20**	**8/35**	**95,50**	**4/15**
VLCS	Dense	68,13	0/17	83,04	0/12	81,36	0/35	82,76	0/15
	LF	66,91	5/17	70,92	1/12	68,87	2/35	**80,24**	6/15
	EGP	70,18	4/17	78,47	6/12	45,85	2/35	—	—
	EASIER	**70,27**	**14/17**	**79,12**	6/12	**78,56**	**4/35**	78,84	6/15
Flowers-102	Dense	88,88	0/17	92,70	0/12	88,50	0/35	86,47	0/15
	LF	77,57	5/17	63,07	4/12	2,86	5/35	87,90	3/15
	EGP	82,06	3/17	87,40	3/12	0,34	2/35	—	—
	EASIER	**83,43**	**6/17**	**88,89**	**5/12**	**88,37**	**10/35**	**88,32**	3/15
DTD	Dense	60,53	0/17	67,50	0/12	64,41	0/35	64,20	0/15
	LF	59,99	2/17	37,98	4/12	4,89	5/35	63,56	3/15
	EGP	59,10	2/17	60,21	5/12	2,13	2/35	—	—
	EASIER	**62,02**	**3/17**	**62,23**	5/12	**63,83**	**6/35**	**63,62**	**4/15**
Aircraft	Dense	73,36	0/17	76,39	0/12	73,36	0/35	75,85	0/15
	LF	67,60	2/17	44,76	4/12	4,98	4/35	**70,48**	6/15
	EGP	69,04	2/17	73,27	5/12	0,99	2/35	—	—
	EASIER	**70,33**	2/17	**74,44**	**7/12**	**72,55**	4/35	69,70	6/15

4.1 Experimental Setup

We cover a variety of setups by evaluating our method on four popular models: ResNet-18, MobileNet-V2, Swin-T and VGG-16, trained on seven datasets: CIFAR-10 [25], Tiny-ImageNet [27], PACS and VLCS from DomainBed [16], as well as Flowers-102 [35], DTD [10], and Aircraft [33]. All the hyperparameters, augmentation strategies, and learning policies are provided in Appendix, mainly following [29] and [36]. For ResNet-18, MobileNetv2, and VGG-16 all the ReLU-activated layers are taken into account. For Swin-T, all the GELU-activated layers are considered. Moreover, the threshold δ is established for each dataset and architecture pair to enable a fair comparison with the existing LF and EGP approaches in terms of top-1 performance with a comparable number of removed layers.[3]

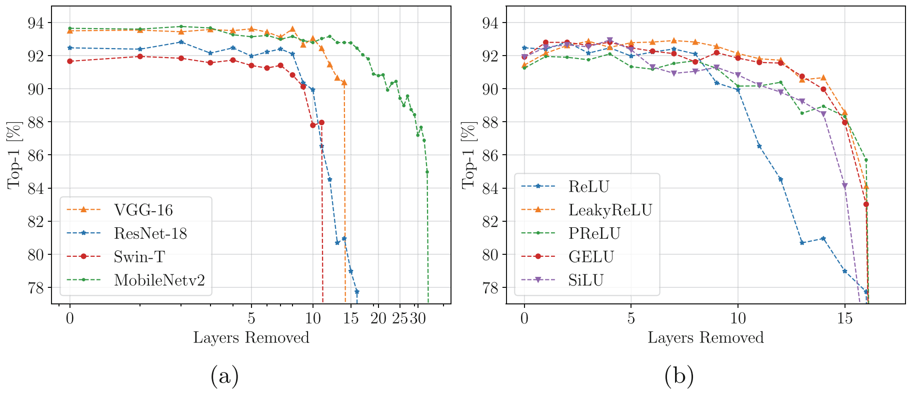

Fig. 3. (a) EASIER applied on ResNet-18, VGG-16, Swin-T and MobileNetv2 networks on CIFAR-10. For each model, we gradually remove non-linear layers. (b) EASIER applied on ResNet-18 on CIFAR-10 with different rectifiers: ReLU, LeakyReLU, PReLU, GELU, and SiLU. Our method is not bound to a specific one and is effective with the most popular.

4.2 Results

A First Overview. We first test our method on a widely known dataset: CIFAR-10. Figure 3a shows the test performance (Top-1) versus the number of removed layers for all the considered models on CIFAR-10, achieved with our method EASIER. Interestingly, all the models exhibit a similar depth-accuracy trend, regardless of their initial depth. Indeed, they first all preserve their original performance, until it drops significantly once ten or so layers have been removed.

Table 1 shows the test performance (top-1) as well as the number of removed layers (Rem.) for all the considered setups. For each combination of dataset and architecture, the performances obtained for each iteration are shown in the tables in the Appendix.

[3] The code and the Appendix are available at https://github.com/VGCQ/EASIER.

Concurrent Method Failure in Some Setups. First, we highlight that the results for EGP on the VGG-16 architecture are not reported apart from CIFAR-10. Indeed, the EGP technique suffers from the layer collapse phenomenon [42]: by forcing a layer to have a zero-entropy, it could force it to be always in the OFF region of its activation, hence preventing the signal from passing through this layer, and therefore leading to a complete failure of the algorithm. This is what is happening on CIFAR-10, where a whole layer is pruned. Since EGP is not working with the VGG architecture on this dataset, we choose not to run the experiments for VGG on other datasets to save computations. Nonetheless, this is not the case with other architectures like ResNet-18, Swin-T, and MobileNetv2, which all have skip connections, leaving another alternative for the signal to pass from the input to the output, in the case the full layer is pruned. However, we also report a problem with transfer learning tasks (Flowers-102, DTD, and Aircraft) for the MobileNetv2 architecture. Indeed, from the first iteration, the EGP's pruning mechanism focuses on the last single layer before the classifier head, leading to its complete removal and hence observing the same layer collapse phenomenon given the absence of a skip/residual connection at this point.

Moreover, even if the results are reported in the table, we underline the failure of Layer Folding for MobileNetv2 in transfer learning setups (Flowers-102, DTD, and Aircraft). The employed auxiliary loss that encourages activations to become linear appears to have a strong effect on the final loss function. The hyperparameter balancing this regularization plays a critical role: a high value prioritizes depth reduction at a cost of performance degradation whereas a small value leads to high performance but with no layers removed. For the mentioned transfer learning task, a trade-off allowing a comparison with EASIER has not been found. This is illustrated by the results obtained on Flowers-102: even with half as many layers removed, LF achieves mediocre performance.

Comparison with Existing Approaches. On most of the considered setups, we can observe the superiority of our method. Indeed, EASIER consistently produces models with better performance for the same number of layers removed, as observed on all the models trained on Tiny-ImageNet-200. For example, while all the methods are able to remove four layers for ResNet-18 on Tiny-ImageNet-200, EASIER achieves respectively 0,6% and 2,56% higher performance than EGP and LF. Moreover, on some setups, EASIER even achieves better performance than the other competitors with more layers removed. This is the case, for example, for all the models trained on CIFAR-10. For instance, for MobileNetv2 on CIFAR-10, EASIER can remove six more layers with 0,94% top-1 gain compared to EGP, which obtains the second-best performance in this setup.

Nevertheless, we highlight the superiority of Layer Folding in two setups: VGG-16 trained on VLCS and Aircraft, in which the models produced by LF achieve better performance for the same number of layers removed, with performance improvements of 1,4% and 0,78% respectively, compared to EASIER.

Comparison with the Original Model. Although on most setups (such as CIFAR-10) it succeeds in compressing models while maintaining performance similar to the original model, EASIER (but also competing methods) is not capable of compressing models without degrading performance. This is the case, for example, with Swin-T on Tiny-ImageNet-200, which displays a 7% loss compared to the original model. The question of a trade-off between performance and compressibility may therefore arise depending on the model's intended use. Nevertheless, apart from VGG-16 on VLCS and Aircraft, our method produces compressed models with the closest performance to the original model compared with existing methods.

4.3 Ablation Study

In this section, we first perform a study over the used rectifier, showing that our method is not bound to a specific one and is effective with any. Figure 3b shows the test performance of ResNet-18 on CIFAR-10, for different rectifiers versus the number of linearized layers. Our method removes at least 8 layers with a performance improvement for GELU, LeakyReLU, and PReLU and with a marginal performance loss for ReLU and SiLU. We hypothesize that it is due to the presence of more signal in backpropagation for GELU, LeakyReLU, and PReLU. Moreover, to find out whether it was necessary (to maintain good performance) to train the network starting from its previous iteration weights (before a layer linearization), a randomly initialized ResNet-18 with the 8 layers selected by EASIER linearized, was re-trained on CIFAR-10 using the same learning policy. The model achieves a top-1 score of 91,56%, down 0,54% on the performance achieved with EASIER. Despite being costly at training time, we concluded that to maximize the performance of the compressed model, it was important to keep training the model from its previous iteration weights.

Table 2. ResNet-18 on CIFAR-10.

Method	Top-1	Rem.
Dense	92,60	0/17
EASIER 2×	92,26	8/17
EASIER 4×	92,45	8/17
EASIER 8×	91,82	8/17

Furthermore, to clear the way for the design of a one-shot approach, we conduct some experiments directly removing several layers at a time, for example by iteratively linearizing the 2, 4, or 8 layers with the lowest entropy. These approaches are denoted respectively EASIER 2×, EASIER 4×, and EASIER 8×. The results for a ResNet-18 trained on CIFAR-10 are presented in Table 2. For fairness, we report the test performance (Top-1) for an equivalent number of layers removed (Rem.). Hence, for EASIER 2× (respectively 4× and 8×), four (respectively two and one) iterations were necessary to obtain these results.

Despite removing the same number of layers, we observe that EASIER 2× and EASIER 4× yield similar results with a slight drop in performance compared to the original model, while EASIER 8× leads to worse performance. With a performance loss of less than one percent compared to the original model, EASIER 8× raises hope for the design of a one-shot approach, which would be more efficient at training time.

Finally, Table 3 showcases the potential savings in terms of inference time and FLOPs for ResNet-18 on CIFAR-10 on six different devices, including CPUs and GPUs spanning from traditional GPU to embedded devices. In general, the fewer layers the network has, the shorter the inference time and the smaller the number of FLOPs. However, we also observe that blindly removing layers is not sufficient to reduce computation. Indeed, a layer removal can result in an increase in MFLOPs, as observed here at the fifth iteration, which is mainly due to the fusion of two convolutional layers, that can result in a layer having a greater size. For instance, to keep the same input/output ratio, two convolutional layers having a kernel size of 3 will fuse in a convolutional layer having a kernel size of 5. Moreover, looking at inference times, every device shows a different trend. While larger devices, like RTX A4500, show a monotonically decreasing inference time, for smaller devices, like P2000 or Jetson Orin, this is not always the case. We also note the same problem on CPUs, like Raspberry Pi 4, where caching is the major problem when dealing with larger kernels.

4.4 Limitations and Future Work

Despite being a successful approach to alleviating deep neural networks' depth, EASIER also presents some limits, which we discuss below.

Training Efficiency. The iterative nature of our method inevitably leads to a longer training time and more intensive computations to achieve the compressed models, compared for instance, to the Layer Folding approach. However, the increased computational cost of training can be offset by the benefits of using these models for inference. Indeed, since a neural network is going to be used multiple times for inference, it is also important to lessen its computational burden related to this use. As opposed to unstructured pruning which offers very few, if any, practical benefits when it comes to deploying the model in a resource-constrained system, our method reduces the critical path forward propagation undergoes, making it useful for processing on parallel systems like GPUs or TPUs, as the computational demands at inference time are reduced.

Nonetheless, even though the method has been thought out iteratively, there is hope for the design of a one-shot approach, which would be more efficient at training time, as shown by the results discussed in the previous ablation study. Another way to address this problem can be to include the entropy in the minimized objective function. However, this approach is not immediately feasible as it is a non-differentiable metric. Therefore, the exploration of differentiable proxies for the layer's entropy is left as future work.

Table 3. Inference time [ms] and MFLOPs of ResNet-18 on CIFAR-10.

Rem.	MFLOPs	Inference on CPU [ms]		Inference on GPU [ms]			
		Xeon E5-2640	Raspi 4	Jetson Orin	P2000	RTX 2080	A4500
0/17	725,47	13,50	135	8,52	4,45	4,43	3,32
1/17	258,24	9,33	111	8,31	4,53	4,43	3,27
2/17	243,46	9,69	106	7,83	4,28	4,21	3,10
3/17	231,79	9,43	139	7,38	4,02	3,93	2,96
4/17	197,85	10,10	117	6,91	3,79	3,68	2,78
5/17	159,05	11,30	144	6,44	3,60	3,46	2,60
6/17	159,99	8,39	225	6,13	4,11	3,18	1,79
7/17	152,36	9,18	144	6,06	4,16	3,10	1,71
8/17	149,84	9,14	149	6,14	3,67	3,21	1,55

Performance Degradation. It is difficult to compress existing parameter-efficient architectures that are not overfitting, and EASIER cannot decrease the depth of an already underfitting architecture without compromising performance, like for example Swin-T on Tiny-ImageNet-200.

Nevertheless, EASIER was able to demonstrate its superiority over existing methods on all the setups considered. Indeed, for the same number of removed layers, EASIER achieves the best performance or can compress more than existing approaches while maintaining performance. We therefore believe that EASIER is a serious candidate to be considered to achieve this kind of goal.

5 Conclusion

In this work, we have presented EASIER, an entropy-based method for layer withdrawal in rectifier-activated deep neural networks. An entropy-based importance metric has been designed to select layers to remove from the network, aiming at depth reduction while preserving high performance in the considered tasks. The capability and effectiveness of reducing the number of layers in a model of EASIER have been demonstrated by experiments conducted on four popular architectures across seven datasets for image classification. Concerned by the ever-growing AI environmental impact, we hope this work can inspire future optimizations and new ways of thinking about network design.

Acknowledgments. This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement 101120237 (ELIAS). Also, this research was partially funded by Hi!PARIS Center on Data Analytics and Artificial Intelligence. This project was provided with computer and storage resources by GENCI at IDRIS thanks to the grant 2023-AD011013930R1 on the supercomputer Jean Zay's the V100 partition.

References

1. Jaafra, Y., Laurent, J.L., Deruyver, A., Naceur, M.S.: Reinforcement learning for neural architecture search: a review. Image Vis. Comput. **89**, 57–66 (2019)
2. Ali Mehmeti-Göpel, C.H., Disselhoff, J.: Nonlinear advantage: trained networks might not be as complex as you think. In: ICML. PMLR (2023)
3. Baymurzina, D., Golikov, E., Burtsev, M.: A review of neural architecture search. Neurocomputing **474**, 82–93 (2022)
4. Blalock, D., Gonzalez Ortiz, J.J., Frankle, J., Guttag, J.: What is the state of neural network pruning? In: MLSys (2020)
5. Bragagnolo, A., Tartaglione, E., Fiandrotti, A., Grangetto, M.: On the role of structured pruning for neural network compression. In: ICIP. IEEE (2021)
6. Brown, T., et al.: Language models are few-shot learners. In: NeurIPS (2020)
7. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: train one network and specialize it for efficient deployment. In: ICLR (2019)
8. Castillo-Navarro, J., Le Saux, B., Boulch, A., Lefèvre, S.: On auxiliary losses for semi-supervised semantic segmentation. In: ECML PKDD (2020)
9. Cho, M., Joshi, A., Reagen, B., Garg, S., Hegde, C.: Selective network linearization for efficient private inference. In: ICML. PMLR (2022)
10. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., Vedaldi, A.: Describing textures in the wild. In: CVPR (2014)
11. Craig, C.C.: On the frequency function of xy. Ann. Math. Stat. **7**, 1–15 (1936)
12. Cui, G., Yu, X., Iommelli, S., Kong, L.: Exact distribution for the product of two correlated Gaussian random variables. IEEE Signal Process. Lett. **23**(11), 1662–1666 (2016)
13. Dror, A.B., Zehngut, N., Raviv, A., Artyomov, E., Vitek, R., Jevnisek, R.: Layer folding: neural network depth reduction using activation linearization. In: BMVC (2022)
14. Faiz, A., et al.: LLMCarbon: modeling the end-to-end carbon footprint of large language models. In: ICLR (2023)
15. Gale, T., Elsen, E., Hooker, S.: The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574 (2019)
16. Gulrajani, I., Lopez-Paz, D.: In search of lost domain generalization. In: ICLR (2020)
17. Guo, J., et al.: CMT: convolutional neural networks meet vision transformers. In: CVPR (2022)
18. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: NeurIPS. Curran Associates, Inc. (2015)
19. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
20. He, Y., Xiao, L.: Structured pruning for deep convolutional neural networks: a survey. IEEE Trans. Pattern Anal. Mach. Intell **46**(5), 2900–2919 (2023)
21. Hestness, J., et al.: Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409 (2017)
22. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
23. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
24. Jha, N.K., Ghodsi, Z., Garg, S., Reagen, B.: DeepReDuce: ReLU reduction for fast private inference. In: ICML. PMLR (2021)

25. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
26. Kuang, W., Zhu, Q., Li, Z.: Multi-label image classification with multi-scale global-local semantic graph network. In: Koutra, D., Plant, C., Gomez Rodriguez, M., Baralis, E., Bonchi, F. (eds.) Machine Learning and Knowledge Discovery in Databases: Research Track, ECML PKDD 2023. LNCS, vol. 14171, pp. 53–69. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-43418-1_4
27. Le, Y., Yang, X.: Tiny ImageNet visual recognition challenge. CS 231N **7**, 3 (2015)
28. Lee, N., Ajanthan, T., Torr, P.: SNIP: single-shot network pruning based on connection sensitivity. In: ICLR (2019)
29. Liao, Z., Quétu, V., Nguyen, V.T., Tartaglione, E.: Can unstructured pruning reduce the depth in deep neural networks? In: ICCV (2023)
30. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2018)
31. Liu, Z., et al.: Swin transformer: hierarchical vision transformer using shifted windows. In: ICCV (2021)
32. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. In: ICLR (2018)
33. Maji, S., Kannala, J., Rahtu, E., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. Technical report (2013)
34. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909 (2015)
35. Nilsback, M.E., Zisserman, A.: Automated flower classification over a large number of classes. In: Indian Conference on Computer Vision, Graphics and Image Processing, December 2008
36. Quétu, V., Tartaglione, E.: DSD^2: can we dodge sparse double descent and compress the neural network worry-free? In: AAAI (2024)
37. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
38. Real, E., et al.: Large-scale evolution of image classifiers. In: ICML. PMLR (2017)
39. Seijas-Macías, A., Oliveira, A.: An approach to distribution of the product of two normal variables. Discussiones Mathematicae Probab. Stat. **32**, 87–99 (2012)
40. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
41. Sun, D., Wang, M., Li, A.: A multimodal deep neural network for human breast cancer prognosis prediction by integrating multi-dimensional data. IEEE/ACM Trans. Comput. Biol. Bioinform. (2019)
42. Tanaka, H., Kunin, D., Yamins, D.L., Ganguli, S.: Pruning neural networks without any data by iteratively conserving synaptic flow. In: NeurIPS (2020)
43. Tartaglione, E., Bragagnolo, A., Fiandrotti, A., Grangetto, M.: Loss-based sensitivity regularization: towards deep sparse neural networks. Neural Netw. **146**, 230–237 (2022)
44. Tartaglione, E., Bragagnolo, A., Odierna, F., Fiandrotti, A., Grangetto, M.: SeReNe: sensitivity-based regularization of neurons for structured sparsity in neural networks. IEEE Trans. Neural Netw. Learn. Syst. **33**(12), 7237–7250 (2021)
45. Touvron, H., et al.: LLaMA: open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
46. Wang, R., Cheng, M., Chen, X., Tang, X., Hsieh, C.J.: Rethinking architecture selection in differentiable NAS. In: ICLR (2020)

47. Yang, Y., Li, M., Meng, B., Huang, Z., Ren, J., Sun, D.: Rethinking the misalignment problem in dense object detection. In: Amini, M.R., Canu, S., Fischer, A., Guns, T., Kralj Novak, P., Tsoumakas, G. (eds.) Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2022. LNCS, vol. 13715, pp. 427–442. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-26409-2_26
48. Zhu, M.H., Gupta, S.: To prune, or not to prune: exploring the efficacy of pruning for model compression (2018)
49. Zoph, B., Le, Q.: Neural architecture search with reinforcement learning. In: ICLR (2016)