Till the Layers Collapse: Compressing a Deep Neural Network Through the Lenses of Batch Normalization Layers

Zhu Liao, Nour Hezbri, Victor Quétu, Van-Tam Nguyen, Enzo Tartaglione

LTCI, Télécom Paris, Institut Polytechnique de Paris, France
{zhu.liao, victor.quetu, van-tam.nguyen, enzo.tartaglione}@telecom-paris.fr
{nour.hezbri}@ensae.fr

Abstract

Today, deep neural networks are widely used since they can handle a variety of complex tasks. Their generality makes them very powerful tools in modern technology. However, deep neural networks are often overparameterized. The usage of these large models consumes a lot of computation resources. In this paper, we introduce a method called **T**ill the **L**ayers **C**ollapse (TLC), which compresses deep neural networks through the lenses of batch normalization layers. By reducing the depth of these networks, our method decreases deep neural networks' computational requirements and overall latency. We validate our method on popular models such as Swin-T, MobileNet-V2, and RoBERTa, across both image classification and natural language processing (NLP) tasks.

Code — https://github.com/ZhuLIAO001/TLC
Extended version — https://arxiv.org/abs/2412.15077

1 Introduction

Deep neural networks (DNNs) have grown considerably in recent decades, with applications in many different tasks. DNNs capture subtle patterns effectively, enabling a wide range of applications. This also allows them to achieve high accuracy. Their applications cross various domains, including image classification (Barbano et al. 2022), semantic segmentation (Chaudhry et al. 2022), object detection (Carion et al. 2020), natural language processing (Touvron et al. 2023), and the multi-modal tasks (Sun, Wang, and Li 2019). The ability of DNNs to scale with the size of models and datasets has been well-demonstrated (Hestness et al. 2017).

However, while DNNs have shown their scalability, modern DNNs can consist of millions to billions of parameters, which means that the number of floating point operations (FLOPs) required for a single inference is enormous. Not only does this require a lot of computing power, but it also creates huge energy consumption and environmental problems. For instance, models like GPT-3 (Brown et al. 2020), which contains 175 billion parameters, have a huge carbon footprint during training, that emphasizes the need for more sustainable AI.

With growing awareness of AI's environmental impact, there are increasing calls for balance. High performance must align with environmental friendliness. This has led to the rise of model compression techniques. They reduce network size and complexity without impacting performance significantly. Techniques such as pruning (Lee, Ajanthan, and Torr 2019; Tartaglione et al. 2022), which eliminates less critical neurons or weights, and quantization (Han, Mao, and Dally 2015), which reduces the precision of weights and activations, have been instrumental in this regard. Furthermore, Knowledge distillation (Hinton, Vinyals, and Dean 2015) enables transferring knowledge from large, complex models to smaller, efficient ones.

However, most compression techniques focus on reducing parameters and filters. Few address reducing the model's depth. Eliminating parameters or filters has relatively little impact on modern computational resources such as GPUs. Indeed, due to the parallel nature of the computation, the size of layers is mainly limited by memory cache and core availability. The main computational bottleneck is the critical path that the forward propagation must pass through (Ali Mehmeti-Göpel and Disselhoff 2023). We would like to specifically minimize this.

This paper addresses this challenge by introducing a novel method, **T**ill the **L**ayers **C**ollapse (TLC), which compresses DNNs looking through the lenses of batch normalization layers. By leveraging batch normalization parameters, TLC identifies and removes less important layers, thereby decreasing computational demands and latency without significantly compromising model performance. Indeed, in rectifier-activated networks, if the standardized signal is mainly positive, we will know that a linear activation would introduce a minimal error during the forward pass. Conversely, a mainly negative signal leads to outputs close to zero. Leveraging this, we can linearize (or remove) layers in the target model that will minimally alter the model's output. We empirically validate our approach across image classification and natural language processing (NLP) tasks. It maintains accuracy while improving efficiency.

We summarize, here below, our key messages and contributions.

- We propose a method for evaluating the importance of layers (Sec. 3.3) based on the value of the batch normalization parameters (Sec. 3.2).
- We propose TLC, a method that identifies and removes redundant layers by leveraging batch normalization pa-

rameters (Sec. 3.4).

- TLC is tested across various architectures and datasets. It achieves a balance between reducing layers and maintaining performance (Sec. 4.2).

2 Related Works

2.1 Neural Network Depth Reduction

Researchers have been exploring ways to make neural networks shallower without losing their effectiveness. (Chen and Zhao 2019) proposed a layer-wise pruning method based on feature representations to shallow deep neural networks, and then retraining the network using knowledge distillation, aiming to reduce network complexity while maintaining performance. This work stated the possibility of designing a layer-based pruning algorithm. (Ali Mehmeti-Göpel and Disselhoff 2023) introduced a channel-wise method to reduce non-linear units while maintaining similar performance. Moreover, (Dror et al. 2021) proposed a method, Layer Folding (LF), which learns whether non-linear activations can be removed, allowing the folding of consecutive linear layers into one. More specifically, ReLU-activated layers are replaced with PReLU activations and become regularized slopes. During post-training, nearly linear PReLUs are removed, and layers are folded. Unlike these previous methods, which focus on the activation function level to decide whether it should be linear or non-linear, or analyzing at the feature level to assess the necessity of neurons. Our approach, TLC, directly evaluates the importance of layers and retains only the most essential ones.

(Liao et al. 2023) proposed Entropy-Guided Pruning (EGP), which aims to remove entire layers, this method reduces network depth by prioritizing low-entropy layers for pruning. This method targets layers that are less active and removes them entirely while trying to keep the network's performance stable. In the same area, (Quétu, Liao, and Tartaglione 2024) introduced EASIER, a method using entropy-based importance to reduce network depth. Specifically, EASIER evaluates the importance of different layers within the network and selectively retains the critical layers, thereby simplifying the network structure. Unlike EGP, which uses unstructured pruning to gradually induce removable layers, often requires multiple training iterations to remove a single layer. And unlike EASIER, which removes one layer after each training. Our approach attempts to remove multiple layers after each training. This provides our method with a clear advantage in training efficiency over EGP and EASIER.

2.2 Layer's Importance Evaluation

The evaluation of the layer's importance has become a crucial aspect of model compression, particularly in the last decade. (Han et al. 2015) proposed a Weight Magnitude-Based method that assesses neuron importance by analyzing the magnitude of weights. The rationale is that neurons with smaller weights contribute less to the model's output and can be pruned with minimal impact. However, this approach often requires extensive retraining to regain the accuracy lost due to pruning. (Molchanov et al. 2016) evaluate the importance of neurons by leveraging gradient information. More specifically, they select neurons to be pruned by using the first-order Taylor expansion to approximate the change in the loss function to estimate layers' importance. This method still faces challenges in identifying optimal pruning strategies, especially in very deep networks.

Despite significant progress in layer importance evaluation, balancing complexity reduction and performance remains challenging. Our TLC method aims to address this challenge by providing an effective layer-evaluating method.

2.3 Other BatchNorm-based Pruning Strategies

Prior studies have used batch normalization statistics to determine filter significance in CNNs for pruning decisions. For instance, in (Liu et al. 2017), the scaling parameters of batch normalization layers are used to define a sparsity-inducing penalty during training. After training, these scaling parameters are employed again to identify unimportant channels in the network. (Oh et al. 2022) employs the parameters of batch normalization layers to characterize the pre-activation Gaussian distributions of filters under the assumption of a sufficiently large batch size. Filter importance is measured by the expected absolute activation values. This metric is then used to rank filters, with a specific pruning ratio assigned to each layer based on the degradation in performance caused by pruning the layer in the pre-trained model.

While our approach shares some similarities with previous works, it differs in two primary aspects. Firstly, our strategy heavily relies on the behavior introduced by the rectifier nonlinearity, and we do not provide a full ranking of filters using batch normalization layers. Instead, we adopt a strict, hard binary ranking (i.e., *ON/OFF* state). Although the pruned neurons within layers may overlap with our *OFF*-state neurons, the *ON*-state neurons in our case are linearized, whereas in (Oh et al. 2022) the neurons remain untouched. Secondly, the pruning ratio in (Oh et al. 2022) is defined according to the sensitivity of the accuracy to pruning, whereas in our approach, the layer importance metric is defined by the degradation caused by removing the entire layer, which is then leveraged to obtain a ranking of layer removing.

3 Till the Layers Collapse

In this section, we introduce our method TLC, which removes complete layers and reduces the depth of the deep neural network with minimal impact on model performance. We begin by formulating the problem in Sec. 3.1 and conducting an error analysis in Sec. 3.2 to explain the motivation behind our approach. Following, in Sec. 3.2, we outline our surgical layer removal process which leverages the parameters of batch normalization layers. Next, in Sec. 3.3, we provide an importance ranking of the layers, which will guide the removal of the least significant layers using the aforementioned strategy. Finally, in Sec. 3.4, we give an overview of the complete pipeline of our method.

3.1 Problem Formulation

Let us consider a DNN consisting of L layers. Batch norm layers are associated with these neurons such that, in each

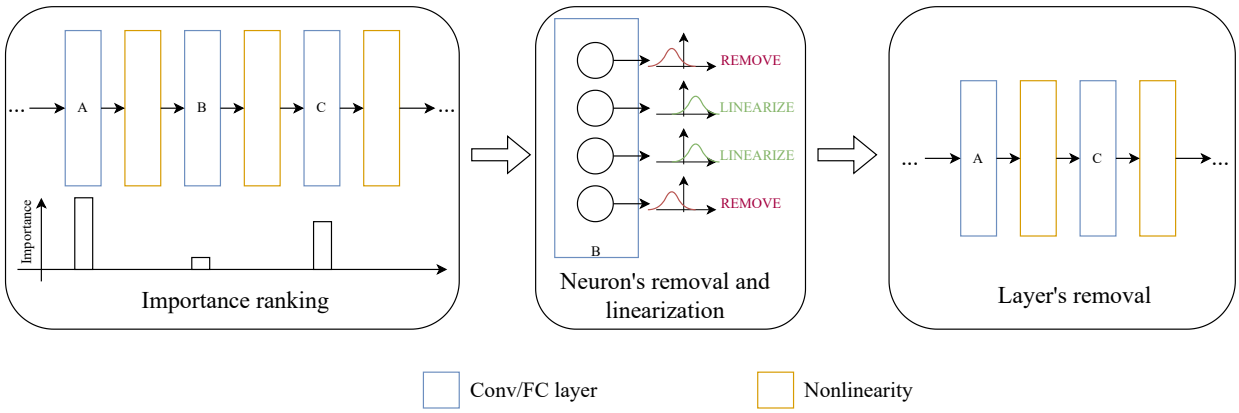

Figure 1: Overview of the key steps for TLC: identification of layer to remove, removal of irrelevant channels, and linearization of the remaining, removal of the layer.

layer, batch normalization is sandwiched between the affine transformation and the nonlinearity. Even in architectures lacking batch norm layers (e.g., transformers), parameters can be estimated via forward propagation and attached.

For the i-th neuron of the l-th layer, let $x_{l,i}$ denote the output of the affine transformation (e.g., convolution). The batch norm layer applies the following transformation:

$$\hat{x}_{l,i} = \frac{x_{l,i} - \mu^B_{l,i}}{\sqrt{(\sigma^B_{l,i})^2 + \epsilon}}, \qquad z_{l,i} = \gamma_{l,i}\hat{x}_{l,i} + \beta_{l,i}, \quad (1)$$

where $\mu^B_{l,i}$ and $\sigma^B_{l,i}$ denote the mean and standard deviation of $x_{l,i}$, and ϵ is an arbitrarily small constant. As we know, $\hat{x}_{l,i} \sim \mathcal{N}(0,1)$. $\gamma_{l,i}$ and $\beta_{l,i}$ are learnable parameters that respectively represent the mean and standard deviation of the batch normalization's output: this means that $z_{l,i} \sim \mathcal{N}(\beta_{l,i}, \gamma^2_{l,i})$.

The bach norm layer is followed by an activation function, where rectifiers are typically used in modern deep neural networks. We denote the rectifier as $\psi_l, \forall l \in [1, L-1]$.[1] Finally, the output of the neuron is:

$$y_{l,i} = \psi_l(z_{l,i}). \quad (2)$$

The distinct feature of rectifiers is that they divide the input space into two regions, with mainly a separate linear function governing each region. The first linear region is usually where the neuron's output is maintained, or very close as for example in GeLU. If the input of the rectifier is in this region, we regard this neuron as at *ON* state. The second is where the rectifier's output of the i-th neuron is asymptotically zero or negative, but with the output's magnitude being lower for the same input magnitude, as for example in LeakyRelu. This neuron is regarded as at *OFF* state.

To effectively collapse layers, it is important to take into account the different influences of different neurons. Therefore, we propose to dissect the behavior of the layers by analyzing individual neurons, rather than considering the layers as a whole. This refinement approach will be guided by the statistics provided.

[1]Please note that the output layer typically has a different nonlinearity-and besides, it is a layer that can not be removed.

3.2 Effect of the Layer Removal

To reduce the depth of DNNs, several works studied the possibility of collapsing the nonlinearity (Dror et al. 2021). When collapsing the nonlinearity, a unified behavior across neurons is enforced, forcefully shifting all the neurons to one side of the rectifier. Herein, we focus on the implications for the neurons when collapsing the nonlinearity, and we aim to determine the error introduced by this process by examining it at the neuron level using the summary statistics provided by the batch normalization layers.

When we remove a neuron using TLC, we define the likelihood of the error $\mathcal{E}_{l,i}$ as the probability of a neuron's state being regarded mistakenly. When the rectifier ψ_l is substituted with the identity function, this is equivalent to shifting all the neurons of the l-th layer to the always *ON* state, we call it always *ON*. Conversely, substituting ψ_l with the null function pushes all the neurons to the always *OFF* state, we call it always *OFF*. So, the error likelihood for the i-th neuron is:

$$\mathcal{E}_{l,i} = \Phi\left(-\frac{|\beta_{l,i}|}{\gamma_{l,i}}\right), \quad (3)$$

where Φ denotes the cumulative distribution function of $\mathcal{N}(0,1)$. In Fig. 2, the blue curve shows how a neuron's error likelihood varies with $\beta_{l,i}$ when substituting the rectifier with an identity function, and the orange curve shows the error likelihood when substituting the rectifier with a null function. Obviously, if all neurons' rectifiers in a layer are substituted uniformly, in either case, there will be unacceptable probability of error.

Based on this analysis, we devise a layer-removing scheme, in which we selectively linearize the neurons and reconfigure the layers, conditioned on the batch norm layer's parameters.

Specifically, inside the layer l, we discriminate between

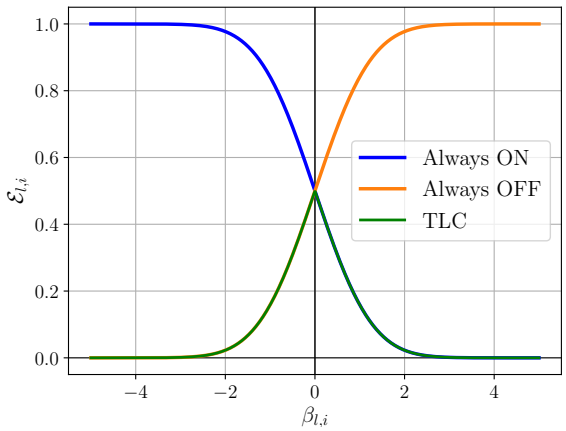

Figure 2: Error plot for the i-th neuron of the l-th layer as a function of the batch norm mean parameter $\beta_{l,i}$ for a standard deviation $\gamma_{l,i} = 1$.

the *ON* and *OFF* states of the neurons leveraging the computed statistics of the BatchNorm layer $\beta_{l,i}, \forall i \in [1, N_l]$.

- When $\beta_{l,i} \leq 0$, the i-th neuron is more likely to be in the *OFF* state, its contribution is therefore marginal and we suppose it can be removed from the network.
- When $\beta_{l,i} > 0$, the i-th neuron is more likely to be in the *ON* state; it is linearized and merged within the subsequent layer via linear combination similarly to (Pilo et al. 2024) to account for its contribution.

This scheme strikes a balance between two extremes: completely shutting the layer (i.e., always *OFF* state) and fully linearizing the activation function (i.e., always *ON* state). By adopting this approach, we effectively minimize the error introduced by these two limiting scenarios. As shown in Fig. 2, the overall error likelihood incurred by our scheme (represented by the area under the green curve) is significantly smaller than that resulting from either of the two extreme cases. This demonstrates our approach can remove layers while minimizing the impact on performance.

Please note that in the transformer architectures we adopt in the article, LayerNorm exists before the fully-connected layer, no normalization was implemented between the layer and the activation. In this case, LayerNorm parameters were unused. Instead, we calculated the average and standard deviation at the fully connected layer's output. Based on the computed average, we decide whether to set each neuron to *OFF* or *ON*.

3.3 Layers' Importance Ranking

In TLC, we shall remove layers based on the importance ranking we adopt, mainly conditioned on the change in performance the removing of a layer from a complete pre-trained model would engender.

Starting from a pre-trained model M, we remove the l-th layer with the method we mentioned in Sec. 3.2. The performance of the pruned model $M_{\text{rem}\{l\}}$ is then evaluated.

Algorithm 1: Our proposed method TLC.

1: **function** TLC(M, $\mathcal{D}_{\text{TRAIN}}$, $\mathcal{D}_{\text{VAL}}$, θ)
2: $M \leftarrow \text{Train}(M, \mathcal{D}_{\text{train}})$
3: $\mathcal{A}^{\text{init}} \leftarrow \text{Evaluate}(M, \mathcal{D}_{\text{val}})$
4: $M' \leftarrow M$
5: $\mathcal{A}_{M'} \leftarrow \mathcal{A}^{\text{init}}$
6: **while** $\mathcal{A}_{M'} \geq \theta \cdot \mathcal{A}^{\text{init}}$ **do**
7: $M \leftarrow M'$
8: $L \leftarrow$ list of layers in M
9: $L_{\text{ranked}} \leftarrow \text{Rank}(L)$
10: $i \leftarrow 1$
11: **while** $\mathcal{A}_{M_{\text{test}}} \geq \mathcal{A}_{M'}$ **do**
12: $M' \leftarrow M_{\text{test}}$
13: $M_{\text{test}} \leftarrow \text{Remove}(M', L_{\text{ranked}}[i])$
14: $\mathcal{A}_{M_{\text{test}}} \leftarrow \text{Evaluate}(M_{\text{test}}, \mathcal{D}_{\text{val}})$
15: $i \leftarrow i + 1$
16: **end while**
17: $M' \leftarrow \text{Train}(M', \mathcal{D}_{\text{train}})$
18: $\mathcal{A}_{M'} \leftarrow \text{Evaluate}(M', \mathcal{D}_{\text{val}})$
19: **end while**
20: **return** M
21: **end function**

We define the layer importance relation $\mathcal{I}$ between the layers l and l' looking at the model's accuracy $\mathcal{A}$ as follows:

$$\mathcal{I}(l) < \mathcal{I}(l') \Leftrightarrow \mathcal{A}(M_{\text{rem}\{l\}}) < \mathcal{A}(M_{\text{rem}\{l'\}}). \quad (4)$$

Layers within the model do not affect the model's performance in the same way. Thus, the effects of their removal would vary notably. In particular, we expect that removing some layers would result in a drop in performance compared to the original pre-trained model (i.e. $\mathcal{A}(M) - \mathcal{A}(M_{\text{rem}\{l\}}) > 0$). In other cases though, we might even have that the model's accuracy increases(i.e. $\mathcal{A}(M) - \mathcal{A}(M_{\text{rem}\{l\}}) < 0$), which can be attributed for example to overfitting of the full model.

3.4 Overview on TLC

In Alg. 1, we present our method TLC that strategically leverages batch norm layers to prune layers and reduce the depth of DNNs. Given a model M, after vanilla training (line 2), we get model M'. We evaluate its initial accuracy $\mathcal{A}^{\text{init}}$ (line 3) on the validation set $\mathcal{D}_{\text{val}}$, which we use to get an importance ranking of the different layers of the model as in Sec. 3.2 (line 9). Subsequently, we use this ascendent ranking of the layers to guide the layer pruning process, starting with the least important layer, and removing one layer at a time (line 13). For this step, we first entirely remove the neurons whose average pre-activation is negative ($\beta_{l,i} < 0$) and then the remaining are linearized and fused with the next one, according to Fig. 1. The incremental removal of layers redefines each time a new model M_{test}, whose validation accuracy of M_{test} is iteratively evaluated (line 14): if a decrease in the validation accuracy of M_{test} to M' is detected (line 11), the procedure is terminated, and the pruned model M_{test} substitutes M'. Then M' undergoes a retraining process to recover its performance (line 17). The

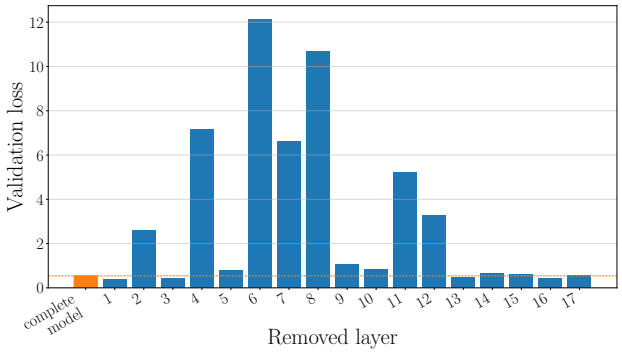

Figure 3: Validation loss for the complete Resnet-18 model pre-trained on Cifar-10 and one layer is removed.

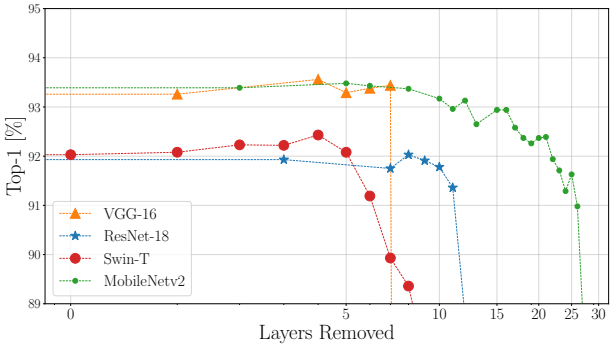

Figure 4: Test performance (top-1) for models trained on CIFAR-10 with different numbers of layers removed by TLC.

final model M is the smallest whose accuracy on the validation set does not drop below a relative threshold θ.

4 Experiments

In this section, we evaluate our method on multiple architectures and datasets for image classification and NLP tasks. All the trainings are performed on an NVIDIA RTX 3090 Ti equipped with 24GB RAM.

4.1 Experimental Setup

We assess our method through image classification and NLP tasks. Concerning image classification, our evaluation encompasses four models: ResNet-18 (He et al. 2016), MobileNet-V2 (Howard et al. 2017), VGG-16bn (Simonyan and Zisserman 2015), and Swin-T (Liu et al. 2021). Models are trained on CIFAR-10 (Krizhevsky, Hinton et al. 2009), Tiny-ImageNet (Le and Yang 2015), ImageNet dataset (Deng et al. 2009), as well as PACS and VLCS from DomainBed (Gulrajani and Lopez-Paz 2020). Training policies follow (Quétu and Tartaglione 2024) and (Xu et al. 2021).

For NLP, our evaluation focuses on two models: BERT (Kenton and Toutanova 2019) and RoBERTa (Liu et al. 2019). Models are trained on SST-2 (Socher et al. 2013), QNLI (Williams, Nangia, and Bowman 2018), and RTE (Bentivogli et al. 2009). We adhere to the training strategies delineated by (Peer et al. 2022) for NLP tasks.

We compare our results with the dense model and two additional baselines: removing layers with the lowest weights/gradients. We also compare our method with existing approaches such as EGP (Liao et al. 2023), Layer folding (LF) (Dror et al. 2021), and EASIER (Quétu, Liao, and Tartaglione 2024).

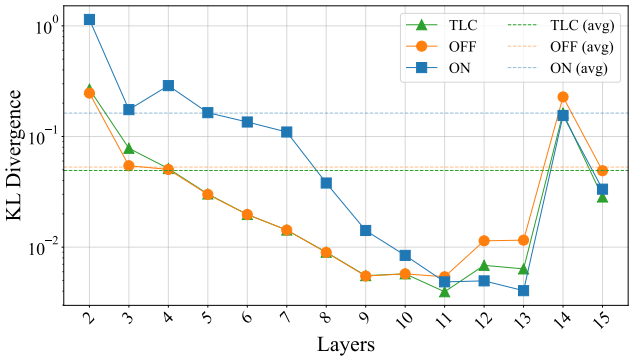

Figure 5: Kullback-Leibler (KL) divergence between the output features of the original VGG-16bn model trained on CIFAR-10 and models removed layers by different methods.

4.2 Results

A first overview. First, we tested our method TLC across different models trained on the CIFAR-10 dataset. Fig. 3 shows the validation loss of the Resnet-18 complete model trained on CIFAR-10 and the loss after one layer is removed. For visualization purposes, we show validation loss here. The orange bar is the complete model's validation loss, the other bars are the validation losses after the corresponding layers are removed. The plot shows that removing certain layers can reduce validation loss (equivalently, increases the validation accuracy).

Fig. 4 shows top-1 validation trends for four models. Performance remains stable until a critical number of layers are removed, then drops significantly (particularly evident in VGG-16).

To estimate the error at the whole layers' scale, we compare TLC with always *OFF* and always *ON* on VGG-16bn trained on CIFAR-10 using KL divergence between the output features of the original model and each method. Fig. 5 shows TLC yields lower KL divergence (0.049) compared to always *ON* (0.163) and always *OFF* (0.053), indicating TLC introduces the least error.

Image classification tasks. Table 1 shows test performance (top-1) and removed layers (Rem.) across all the considered image classification setups. We discover that removing layers with the lowest sum weights/gradients fails for the MobileNet architecture. Starting with the removal of the first layer, this mechanism tends to focus on removing the last single layer before the classifier head, leading to gradient explosion in subsequent training. Moreover, EGP results on the VGG-16bn architecture are reported only for CIFAR-

Dataset	Approach	ResNet-18		Swin-T		MobileNet-V2		VGG-16bn	
		top-1	Rem.	top-1	Rem.	top-1	Rem.	top-1	Rem.
CIFAR-10	Dense model	92.00	0/17	91.63	0/12	93.64	0/35	93.09	0/15
	Smallest weights	88.49	11/17	86.92	3/12	10.00	1/35	90.53	7/15
	Smallest gradients	88.60	11/17	86.96	3/12	10.00	1/35	90.4	7/15
	EGP	90.64	5/17	86.04	6/12	92.22	6/35	10.00	1/15
	LF	90.65	1/17	85.73	2/12	89.24	9/35	86.46	1/15
	EASIER	86.53	11/17	91.25	6/12	92.45	16/35	93.03	7/15
	TLC	$\mathbf{90.91 \pm 0.57}$	**12/17**	$\mathbf{91.98 \pm 0.07}$	**6/12**	$\mathbf{92.97 \pm 0.38}$	**17/35**	$\mathbf{93.61 \pm 0.23}$	**7/15**
Tiny-Inet	Dense model	41.86	0/17	75.88	0/12	45.70	0/35	58.44	0/15
	Smallest weights	37.42	8/17	72.90	1/12	0.5	1/35	56.88	1/15
	Smallest gradients	37.88	8/17	72.92	1/12	0.5	1/35	57.34	1/15
	LF	37.86	4/17	50.54	1/12	25.88	12/35	31.22	1/15
	EGP	37.44	5/17	71.48	1/12	46.88	1/35	—	—
	EASIER	35.84	6/17	70.94	1/12	47.58	11/35	55.16	1/15
	TLC	$\mathbf{38.66 \pm 0.68}$	**9/17**	$\mathbf{74.07 \pm 0.02}$	**1/12**	$\mathbf{47.84 \pm 0.55}$	**16/35**	$\mathbf{57.63 \pm 0.65}$	1/15
PACS	Dense model	79.70	0/17	97.00	0/12	96.10	0/35	96.10	0/15
	Smallest weights	84.30	8/17	95.10	3/12	18.50	1/35	95.20	3/15
	Smallest gradients	83.60	6/17	95.90	3/12	18.50	1/35	95.50	1/15
	LF	82,90	3/17	87,70	2/12	79.70	1/35	93.60	1/15
	EGP	81.60	3/17	93.50	4/12	17.70	3/35	—	—
	EASIER	**88.30**	**9/17**	93.80	3/12	94.40	7/35	95.20	3/15
	TLC	84.80 ± 0.78	9/17	$\mathbf{96.57 \pm 0.41}$	**4/12**	$\mathbf{94.87 \pm 0.19}$	**11/35**	$\mathbf{95.98 \pm 0.22}$	**4/15**
VLCS	Dense model	67.85	0/17	85.83	0/12	81.83	0/35	84.62	0/15
	Smallest weights	65.89	16/17	69.99	5/12	6.43	1/35	80.71	7/15
	Smallest gradients	66.26	11/17	70.18	5/12	6.43	1/35	80.99	7/15
	LF	63.28	7/17	70.92	1/12	68.87	2/35	80.24	2/15
	EGP	64.40	5/17	82.76	3/12	45.85	2/35	—	—
	EASIER	54.24	15/17	78.19	5/12	72.88	22/35	78.84	6/15
	TLC	$\mathbf{66.43 \pm 0.66}$	**16/17**	$\mathbf{82.79 \pm 0.31}$	**5/12**	$\mathbf{76.11 \pm 1.18}$	**23/35**	$\mathbf{81.41 \pm 0.42}$	**7/15**
ImageNet	Dense model	68.28	0/17	81.08	0/12	71.87	0/35	73.37	0/15
	Smallest weights	67.80	2/17	79.74	1/12	0.1	1/35	70.67	1/15
	Smallest gradients	67.56	2/17	79.71	1/12	0.1	1/35	70.12	1/15
	LF	67.62	1/17	73.51	1/12	7.89	1/35	72.22	2/15
	EGP	61.73	2/17	78.62	1/12	0.1	1/35	—	—
	EASIER	67.20	2/17	78.78	1/12	41.14	2/35	1.19	1/15
	TLC	**67.81**	**2/17**	**79.96**	**1/12**	**59.43**	**2/35**	**72.89**	**2/15**

Table 1: Test performance (top-1) and the number of removed layers (Rem.) for all image classification setups considered. The best results between Smallest weights/gradients, LF, EGP, EASIER, and TLC are in **bold**.

10 due to the layer collapse phenomenon: when forcing a layer to have zero entropy, it remains in the *OFF* state; this prevents signal transmission. Accordingly, to save computational resources, we did not train VGG-16bn on other datasets with EGP. However, architectures such as ResNet-18, Swin-T, and MobileNet-V2 do not exhibit this problem due to the presence of skip connections, which provide alternate paths for signal flow even if an entire layer is pruned. Meanwhile, TLC retains enough *ON* state neurons to ensure proper signal transmission. Moreover, TLC avoids removing performance-critical layers. As a result, TLC does not exhibit the problems that removing layers with the lowest sum weights/gradients and EGP encountered, it works well with all considered architectures.

Compared to LF, TLC removes more layers while maintaining or improving top-1 accuracy. Compared to EASIER, TLC achieves comparable (in most cases better) results for models trained on CIFAR-10, Tiny-ImageNet, and PACS, as well as for ResNet-18 and Swin-T models trained on ImageNet. However, for models trained on VLCS, and for MobileNet-V2 and VGG-16bn models trained on ImageNet, TLC consistently yields significantly better top-1 accuracy at the same level of layer removal. Moreover, EASIER is an iterative method, it removes only one layer at a time. Since TLC tries to remove multiple layers together, our method has a significant advantage in training efficiency.

NLP tasks. Table 2 shows the results for all the NLP setups. Similarly to what observed for image classification tasks, TLC can obtain models with layer removal and maintain good performance. The results show that removing the layer with the lowest sum of weights/gradients results performs close to TLC in most setups. Both methods can remove layers from models with minimal or no performance degradation. This reveals the presence of redundancy in these models. It also appears that On NLP tasks, TLC outperforms LF, EGP, and EASIER by achieving higher accuracy, more removable layers, or both. The exception rises for RTE, where in general the number of removable layers is low. We hypothesize that the pre-trained models are not a good fit for this specific downstream task, also looking at a lower performance of the Dense model compared to SST-

Dataset	Approach	BERT		RoBERTa	
		top-1	Rem.	top-1	Rem.
SST-2	Dense model	92.55	0/12	94.04	0/12
	Smallest weights	90.14	3/12	92.20	5/12
	Smallest gradients	90.25	4/12	92.43	4/12
	LF	84.52	2/12	50.92	2/12
	EGP	85.09	4/12	86.47	5/12
	EASIER	84.63	3/12	86.81	4/12
	TLC	$\mathbf{91.44 \pm 0.61}$	**4/12**	$\mathbf{93.00 \pm 0.28}$	**6/12**
QNLI	Dense model	90.61	0/12	91.47	0/12
	Smallest weights	83.65	9/12	79.55	6/12
	Smallest gradients	84.64	10/12	80.41	8/12
	LF	49.46	1/12	50.54	2/12
	EGP	82.85	9/12	84.66	4/12
	EASIER	50.54	3/12	50.54	3/12
	TLC	$\mathbf{84.80 \pm 0.92}$	**10/12**	$\mathbf{89.53 \pm 1.97}$	**8/12**
RTE	Dense model	57.04	0/12	70.40	0/12
	Smallest weights	46.93	1/12	**75.81**	**1/12**
	Smallest gradients	55.23	1/12	72.20	1/12
	LF	52.71	1/12	47.29	1/12
	EGP	57.73	1/12	52.71	1/12
	EASIER	53.07	1/12	47.29	1/12
	TLC	$\mathbf{59.08 \pm 1.68}$	**1/12**	74.13 ± 0.61	1/12

Table 2: Test performance (top-1) and the number of removed layers (Rem.) for all the considered NLP setups.

Activation	Approach	top-1	Rem.
ReLU	Dense model	92.00	0/17
	TLC	91.36	12/17
SiLU	Dense model	92.22	0/17
	TLC	91.72	12/17
PReLU	Dense model	91.55	0/17
	TLC	90.57	12/17
LeakyReLU	Dense model	91.79	0/17
	TLC	92.00	11/17
GELU	Dense model	91.83	0/17
	TLC	91.84	12/17

Table 3: Analysis with different activation functions on ResNet-18 trained on CIFAR-10.

2 or QNLI: for this, removing layers might not be a viable strategy. This raises a warning when employing approaches that reduce the model's depth.

4.3 Ablation Study

Table 3 shows the test performance of ResNet-18 on CIFAR-10, for different rectifiers versus the number of linearized layers. As expected, TLC is compatible with the most common rectifiers, removing a similar amount of layers. The performance gap recorded is due to the different nonlinearities employed. It appears that TLC is not bound to a specific one and is effective with all the most popular choices.

Table 4 presents a measure of FLOPs and real memory occupation, on Swin-T trained on CIFAR-10 with layers collapsed through TLC. Generally, the fewer layers the network has, the smaller the number of FLOPs, and the smaller the memory usage.

4.4 Limitations

TLC is a successful approach to alleviate deep neural networks' computational burden by decreasing their depth.

Rem.	MFLOPs	Mem.usage [MBs]	top-1
0/12	8987.13	115.80	91.63
1/12	8582.51	102.54	92.03
2/12	8177.89	100.35	92.08
3/12	7773.27	83.40	92.23
4/12	7368.65	67.33	92.22
5/12	6964.03	63.95	92.43
6/12	6559.41	59.44	92.08
7/12	6154.79	58.45	91.19
8/12	5750.18	57.47	89.93

Table 4: MFLOPs and Memory usage [MBs] of Swin-T on CIFAR-10 on NVIDIA RTX 4500.

Model	Approach	top-1	Rem.
ResNet-18	Dense model	92.00	0/17
	TLC-finetuning	91.12	7/17
Swin-T	Dense model	91.63	0/12
	TLC-finetuning	89.49	2/12
MobileNet-V2	Dense model	93.64	0/35
	TLC-finetuning	92.52	15/35
VGG-16bn	Dense model	93.09	0/15
	TLC-finetuning	92.08	8/15

Table 5: Analysis for models trained on CIFAR-10 dataset and pruned by the TLC-finetuning method.

Meanwhile, we also notice that TLC leads to long training times and high computational requirements.

To reduce training costs, we propose TLC-finetuning, an approach with a shorter fine-tuning process that focuses on the final training stage. We tested the TLC-finetuning method on different models with the CIFAR-10 dataset. As shown in Table 5, although the ability to remove layers is not as significant as the method that involves full retraining in each iteration, TLC-finetuning still produces models that retain good top-1 performance while allowing for layer removing. It appears that our method can be scaled to larger language models and remains effective in more complex scenarios. We leave further exploration and refinement of this approach for future research.

5 Conclusion

In this work, we have presented TLC, a method designed to reduce the depth of DNNs efficiently. By utilizing the parameters from batch normalization layers, TLC can identify and remove less critical layers while maintaining a good model performance. Our experiments across multiple image classification and NLP tasks demonstrate the robustness and effectiveness of TLC compared to existing methods.

TLC is a step forward in the quest for more sustainable and efficient neural networks, and we hope that in the future we will find even more efficient and environmentally friendly AI.

Acknowledgments

This work was supported by several funding bodies. Part of the work was funded by the Hi!PARIS Center on Data Analytics and Artificial Intelligence. It also received support from the European Union's HORIZON Research and Innovation Programme under grant agreement No. 101120657, as part of the ENFIELD project (European Lighthouse to Manifest Trustworthy and Green AI). Additionally, funding was provided by the French National Research Agency (ANR) under grant agreements ANR-22-PEFT-0003 and ANR-22-PEFT-0007, as part of the France 2030 initiative, specifically the NF-NAI and NF-FITNESS projects. The project also received funding from the European Union's Horizon Europe Research and Innovation Programme under grant agreement No. 101120237 (ELIAS). Zhu Liao acknowledges financial support from the China Scholarship Council (CSC).

References

Ali Mehmeti-Göpel, C. H.; and Disselhoff, J. 2023. Nonlinear Advantage: Trained Networks Might Not Be As Complex as You Think. In *ICML*.

Barbano, C. A.; Tartaglione, E.; Berzovini, C.; Calandri, M.; and Grangetto, M. 2022. A Two-Step Radiologist-Like Approach for Covid-19 Computer-Aided Diagnosis from Chest X-Ray Images. In *ICIAP*.

Bentivogli, L.; Clark, P.; Dagan, I.; and Giampiccolo, D. 2009. The Fifth PASCAL Recognizing Textual Entailment Challenge. *TAC*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *NeurIPS*.

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-end object detection with transformers. In *ECCV*.

Chaudhry, H. A. H.; Renzulli, R.; Perlo, D.; Santinelli, F.; Tibaldi, S.; Cristiano, C.; Grosso, M.; Fiandrotti, A.; Lucenteforte, M.; and Cavagnino, D. 2022. Lung Nodules Segmentation with DeepHealth Toolkit. In *ICIAP*.

Chen, S.; and Zhao, Q. 2019. Shallowing Deep Networks: Layer-Wise Pruning Based on Feature Representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*.

Dror, A. B.; Zehngut, N.; Raviv, A.; Artyomov, E.; Vitek, R.; and Jevnisek, R. 2021. Layer folding: Neural network depth reduction using activation linearization.

Gulrajani, I.; and Lopez-Paz, D. 2020. In Search of Lost Domain Generalization. In *ICLR*.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both Weights and Connections for Efficient Neural Network. In *NeurIPS*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Hestness, J.; Narang, S.; Ardalani, N.; Diamos, G.; Jun, H.; Kianinejad, H.; Patwary, M. M. A.; Yang, Y.; and Zhou, Y. 2017. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Kenton, J. D. M.-W. C.; and Toutanova, L. K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Le, Y.; and Yang, X. 2015. Tiny imagenet visual recognition challenge. *CS 231N*.

Lee, N.; Ajanthan, T.; and Torr, P. 2019. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*.

Liao, Z.; Quétu, V.; Nguyen, V.-T.; and Tartaglione, E. 2023. Can Unstructured Pruning Reduce the Depth in Deep Neural Networks? In *ICCV*.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning Efficient Convolutional Networks through Network Slimming. In *ICCV*.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*.

Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; and Kautz, J. 2016. Pruning Convolutional Neural Networks for Resource Efficient Inference. In *ICLR*.

Oh, J.; Kim, H.; Baik, S.; Hong, C.; and Lee, K. M. 2022. Batch normalization tells you which filter is important. In *WACV*.

Peer, D.; Stabinger, S.; Engl, S.; and Rodríguez-Sánchez, A. 2022. Greedy-layer pruning: Speeding up transformer models for natural language processing. *Pattern Recognition Letters*.

Pilo, G.; Hezbri, N.; Pereira e Ferreira, A.; Quétu, V.; and Tartaglione, E. 2024. Layerfold: A Python Library to Reduce the Depth of Neural Networks.

Quétu, V.; Liao, Z.; and Tartaglione, E. 2024. The simpler the better: An entropy-based importance metric to reduce neural networks' depth. In *ECML PKDD*.

Quétu, V.; and Tartaglione, E. 2024. DSD^2: Can We Dodge Sparse Double Descent and Compress the Neural Network Worry-Free? In *AAAI*.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Sun, D.; Wang, M.; and Li, A. 2019. A Multimodal Deep Neural Network for Human Breast Cancer Prognosis Prediction by Integrating Multi-Dimensional Data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*.

Tartaglione, E.; Bragagnolo, A.; Fiandrotti, A.; and Grangetto, M. 2022. Loss-based sensitivity regularization: towards deep sparse neural networks. *Neural Networks*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Williams, A.; Nangia, N.; and Bowman, S. R. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL HLT*.

Xu, Q.; Zhang, R.; Zhang, Y.; Wang, Y.; and Tian, Q. 2021. A Fourier-Based Framework for Domain Generalization. In *CVPR*.