LayerCollapse: Adaptive compression of neural networks

Soheil Zibakhsh Shabgahi [1] Mohammad Sohail Shariff [1] Farinaz Koushanfar [1]

Abstract

Handling the ever-increasing scale of contemporary deep learning and transformer-based models poses a significant challenge. Overparameterized Transformer networks outperform prior art in Natural Language processing and Computer Vision. These models contain hundreds of millions of parameters, demanding significant computational resources and making them prone to overfitting. In this work we present LayerCollapse , a form of structured pruning to reduce the depth of fully connected layers. We develop a novel regularizer allowing for post-training compression without finetuning, while having limited impact on performance. LayerCollapse controls model expressiveness with regularization on the activations between fully connected layers, modulating the linearity of activation functions. A linear activation function reduces the rank of the transformation to the rank of the corresponding linear transformation. We demonstrate the effectiveness of LayerCollapse by showing its compression capabilities in sentimental analysis and image classification benchmarks. Moreover we show LayerCollapse is an effective compression aware regularization method in a language modeling benchmark.

1. Introduction

The expanding scope of Deep Learning (DL) applications has led to an escalating demand for more accurate models, resulting in an ongoing rise in their complexity. This surge in complexity, however, presents a challenge: It increases the execution costs, in terms of memory and latency. This makes it difficult to deploy these models in real-world scenarios on standard hardware. Recent studies have shown that modern neural networks often contain unnecessary redundancies, which can be eliminated without affecting their performance. This discovery has spurred interest in model compression techniques such as unstructured pruning (Li et al., 2016)(Jiang et al., 2023), structured pruning (Fan et al., 2019), quantization (Hubara et al., 2017)(Jacob et al., 2018)(Nguyen et al., 2020)(Krishnamoorthi, 2018), and model distillation(Hinton et al., 2015). These methods aim to identify and remove these redundant elements effectively. However, despite significant advances in model compression, implementing these optimizations typically requires additional steps such as fine-tuning or retraining. These steps can be resource-intensive, making the compression techniques less practical. Moreover some of these methods like unstructured pruning require support from hardware and software to take advantage of the compressed model (Sundaram et al., 2023).

Table 1. Percentage of model parameters and MACs reductions attainable by collapsing their MLP layers.

Model	% Params	% MACs
GPT2-Large	53.4	52.7
Bert-Large	52.6	57.1
ViT L/16	57.9	58.2
Mixer L/16	86.5	86.5
VGG 16	71.2	0.6

Multi-Layer Perceptrons (MLPs) (Haykin, 1994) are a key architectural component of modern deep neural networks. MLPs are regarded by many as the first contribution to deep learning, present in almost all transformer architectures, responsible for more than half of the total parameters in these models. In its simplest form, a K-layer MLP can be described as K linear transformations separated by some non-linear operations. Equivalently, a MLP can also be described as a sequence of fully connected (a.k.a., dense), layers separated by non-linear layers. MLPs are central to the success of diverse deep learning architectures including encoder only transformer models (Vaswani et al., 2017)(Lan et al., 2019)(Raffel et al., 2020), causal transformer models (Radford et al., 2019)(Touvron et al., 2023)(Gunasekar et al., 2023), Vision Transformers (Dosovitskiy et al., 2020), and MLP-Mixers (Tolstikhin et al., 2021).

Due to the dense nature of MLPs, a significant portion of a model's computational overhead and storage requirements can be attributed to its MLP layers. For instance, MLPs

[1]Department of ECE, University of California San Diego, CA, USA. Correspondence to: Soheil Zibakhsh Shabgahi <szibakhshshabgahi@ucsd.edu>.

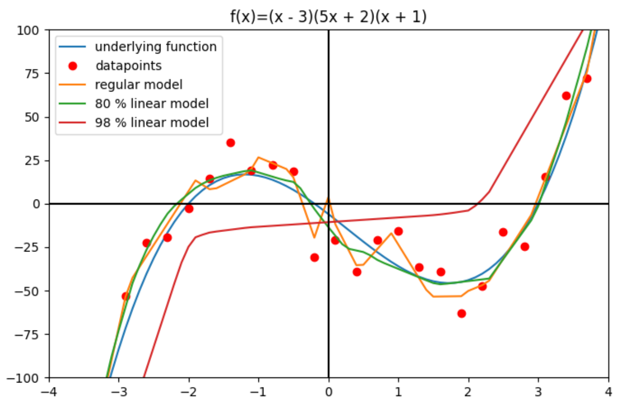

Figure 1. Illustration of regression performance on a two-layer neural network, demonstrating the impact of varying levels of ReLU activation linearity on model overfitting and underfitting. We define the percent of linearity to be the negative slope.

constitute over 60% of the model's total parameters for the popular models discussed in (Dosovitskiy et al., 2020)(Tolstikhin et al., 2021)(Simonyan & Zisserman, 2014).

MLPs contain non-linear activations, e.g., ReLU, which separate sequences of linear transformations. If the non-linear layers were absent, the consecutive linear layers could be condensed into a single layer. To showcase the immense potential of computation and storage reductions when linearizing MLPs, we calculate this reduction in mentioned architectures in table 1. This table demonstrates the reduction percentage in the number of parameters and MAC operations when we replace the MLPs with a linear transform in relevant NLP and vision architectures.

The core thesis of our work revolves around non-linear activations. We argue that the expressiveness of an MLP is primarily governed by its non-linear components. By modulating these non-linear characteristics, we can orchestrate a continuum of model expressiveness, ranging from highly complex to near-linear transformations. Figure 1 demonstrates this idea using a simple regression task and an MLP with 2 layers and a Parametric ReLU (PReLU) as the activation (He et al., 2015). As illustrated, having complex models with lack of a sizable dataset will lead to overfitting. By modulating the negative slope of the activation function one can regulate model's expressiveness.

Common, regularization strategies such as Dropout(Srivastava et al., 2014), ℓ_1 and ℓ_2 norms, have been employed to restrict model expressiveness, attempting robust generalization and preventing overfitting. These regularizers change the model weight structures such that post training compression like pruning and quantization pared with finetuning will yield efficient high performing models. (Fan et al., 2019) introduced LayerDrop, a regularization technique to allow for post training compression without finetuning. Extending this principle, we introduce a LayerCollapse regularization by using the nature of ReLU activations. This regularizer works by modulating the negative slope of ReLU to one, making it a linear transformation. This not only facilitates the fusion of successive linear layers, conserving computational resources and storage, but also can lead to better model generalizability.

This paper presents several key contributions to the field of neural network regularization and compression:

- We introduce LayerCollapse , a new regularization approach that enables effective one-shot compression after training.
- A theoretical analysis provides bounds on the compression loss associated with LayerCollapse .
- We propose a compression-aware regularization loss designed to encourage simpler model structures during the training phase.
- Through experiments across various datasets in both vision and NLP fields, we demonstrate that LayerCollapse achieves post-training compression rates of up to 74%. These experiments also highlight enhanced model generalization and reduced overfitting, alongside the benefits of compression.
- Our work presents a comparative analysis of computational efficiency between knowledge distillation and LayerCollapse, showcasing an improvement of 8% in final results with LayerCollapse , while consuming 80% less computational resources.
- We offer an open-source library for LayerCollapse, facilitating its easy integration with PyTorch frameworks.[1]

2. Related work

In the current state of over parameterized neural networks two major concerns are the point of attention for academia and industry: Deployment of such models to reduce the requirements in terms of compute power and storage, stopping such large models from overfitting. compression techniques aim to reduce computational demands while ensuring model precision. Predominantly, the methods employed for model compression are pruning (Han et al., 2015), quantization (Hubara et al., 2017)(Jacob et al., 2018), and distillation (Hinton et al., 2015), each with its unique approach toward model optimization (Gupta & Agrawal, 2022). The goal of compression is not only the size of the model but also

[1]https://github.com/sohail2810/LayerCollapse

latency and resource utilization. Different methods reach these goals differently. For instance, quantization aims at reducing the number of bits needed to store weights. This reduction doesn't necessarily reduce computation cost as it needs support from the hardware to take advantage of the quantized data structure. Pruning strategies can broadly be categorized as structured or un-structured. In unstructured pruning the storage requirements can reduce since a different sparse representation can be utilized to store and load the weights. However, it is notable that high prune rate, and software support are required to take advantage of this method. (Sundaram et al., 2023) introduce a hardware component to take advantage of unstructured pruning in decreasing inference time. In contrast structured pruning methods (Fan et al., 2019)(Lemaire et al., 2019)(Fu et al., 2022) and Knowledge Distillation (Hinton et al., 2015), change the architecture of the model, such that further hardware and software support is not required to take advantage of the model post compression. In the same light, LayerCollapse changes the architecture of the neural network, thus no further software or hardware optimization is required to take advantage of this method. In section 4 we compare our model with both *LayerDrop* (Fan et al., 2019) and Knowledge distillation (Hinton et al., 2015). We show the advantages and disadvantages of LayerCollapse compared with these baselines.

Pruning and Quantization can be categorized into iterative and one-shot approaches. Iterative methods compress the model in steps, finetuning the model in between. In contrast one-shot methods use some sort of regularization in training, enabling the model to be compressed without the need for further finetuning. To minimize the loss induced by quantization it is standard practice to use quantization aware training (QAT). In short, QAT regularizes the model weights during training to stay in a close range of their quantized value (Nguyen et al., 2020). A big body of work explores diverse strategies to discern which neurons are expendable, including magnitude (Li et al., 2016) and gradient (Liu & Wu, 2019)(Yang et al., 2022) based methods. Another body of work looks into Pruning Aware Training (PAT). In these methods, a regularization term attempts structural sparsity when training (Lemaire et al., 2019)(Jiang et al., 2023)(Wen et al., 2016)(Alvarez & Salzmann, 2016). This reduces the loss in accuracy when pruning the model post training. (Fan et al., 2019) claim *LayerDrop* is not only a PAT, it improves training results as a regularizer. In section 4.1 we evaluate regularization effect of LayerCollapse against *LayerDrop* in a Language modeling benchmark showing the advantage and disadvantage of our method. Explicit regularization methods includeing Dropout (do Santos et al., 2021)(Ghiasi et al., 2018)(Srivastava et al., 2014) and weight decay follow Occam's razor (Schaffer, 2015) principle. They assume the prior knowledge that simpler models are more likely to be robust. In Section 3, we demonstrate the application of this principle in formulating our regularization term. While classical norms influence weight magnitudes to mitigate overfitting, LayerCollapse adjusts the model's depth and structure.

Previous studies in private inference have explored the elimination of non-linearities in neural networks to enhance computational efficiency. Linear operations in private execution can reach the speed of plaintext executions, but non-linear components significantly increase latency (Cho et al., 2022). DeepReDuce introduces a method to incrementally remove ReLU non-linearities along each activation dimension, creating a "thin" ReLU structure for improved private computation efficiency (Jha et al., 2021). Selective Network Linearization (SNL) (Cho et al., 2022) extends this concept by employing PReLUs, and similar to LayerCollapse , it adopts a regularization strategy for ReLU linearization. Our methodology diverges from SNL in several critical areas. Firstly, networks modified by SNL cannot be simplified further, as all activation dimensions require linearization to reduce the model size. Secondly, our approach introduces a theoretical regularizer and presents empirical evidence of LayerCollapse 's regularization effectiveness. We showcase the one-shot compression capabilities of LayerCollapse , demonstrating significant efficiency gains in streamlined models across computer vision and natural language processing domains. Moreover, our approach is general and applicable to reducing the number of layers in plaintext and ciphertext settings. Increase the scope of research further than computer vision. In particular, we show how our method works for reducing the number of connected layers for LLMs where optimizing the size of the network and the number of layers are of significant value.

3. Methodology

This section presents LayerCollapse , a novel regularization technique designed to adaptively compress the model by reducing its depth, possibly improving generalization in the process. Our approach enables the extraction of a shallower model directly, without the need for finetuning.

As illustrated in Figure 2, LayerCollapse involves regularizing the complexity of a layer by linearizing its activation function when training or fine-tuning. This linearization enables the merging of adjacent linear layers into a single layer, thus effectively reducing the model's depth. The resulting model has lower storage, and computation demands without the need for software/hardware support. Contrary to common pruning (Li et al., 2016) and quantization (Hubara et al., 2017) techniques that often necessitate finetuning of the compressed model, LayerCollapse functions as a regularizer during the training or finetuning phase, making one-shot compression possible.

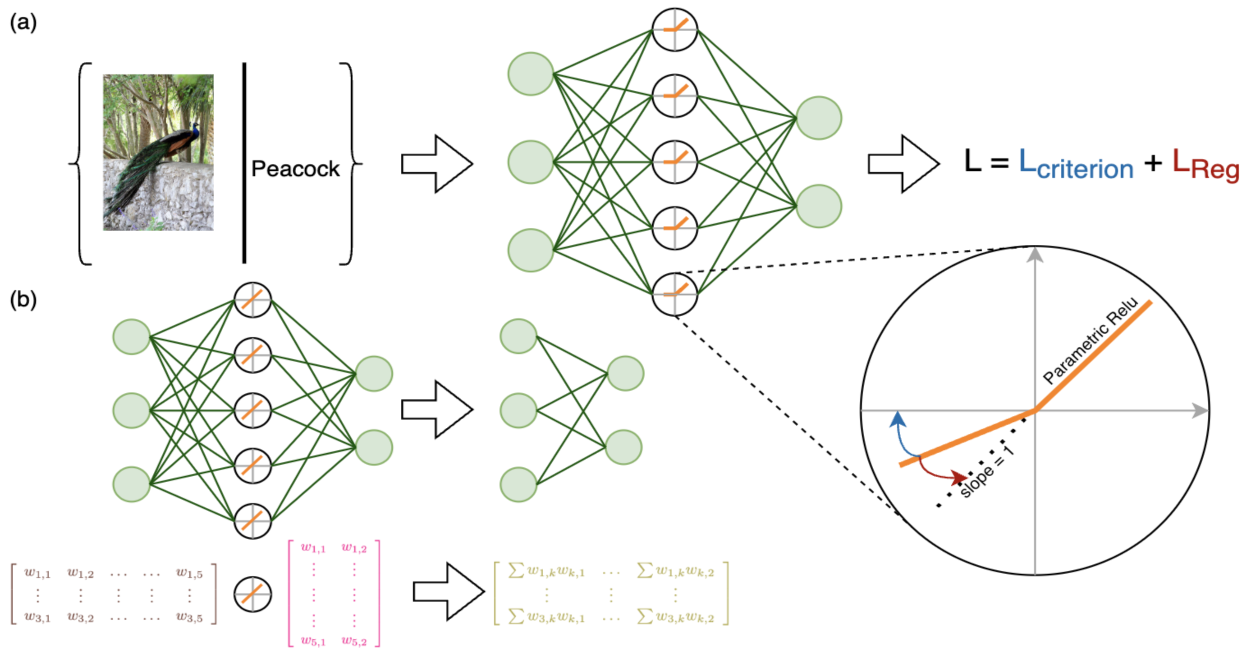

Figure 2. Overview of the LayerCollapse process: (a) The regularization loss shifts the PReLU towards linearity. (b) The layer collapse operation involves eliminating a layer by replacing the hidden layer and adjusting the weights through the product of two matrices.

To utilize LayerCollapse as compression for pre-trained models, a few rounds of fine-tuning with the LayerCollapse regularizer are required to ensure the linearization does not significantly impact accuracy.

We define a Collapsible MLP as: Two consecutive fully connected layers utilizing a Parametric Rectified Linear Unit (PReLU) (He et al., 2015) as the activation. The Parametric Rectified Linear Unit (PReLU) can be expressed as:

$$PReLU_{\alpha}(x) = \max(0, x) + \alpha \min(0, x) \tag{1}$$

where α represents the slope of the PReLU function when x is negative. Output of a basic two-layer collapsible MLP is described as:

$$Y_{\alpha} = W_2(PReLU_{\alpha}(W_1 X + b_1)) + b_2 \tag{2}$$

where, X is the input random variable and Y is the output random variable. W_1 and W_2 are the weight matrices and b_1 and b_2 are the bias vectors for the first and second fully connected layers, respectively. α can be used to make the activation linear. We define Y_{linear} as:

$$Y_{linear} = W_2 W_1 X + W_2 b_1 + b_2 \tag{3}$$

We define the error as the squared difference of Y_{α} and Y_{linear}. In Appendix A, we show that $\forall \delta > 0$

$$P\left\{|Y_{\text{linear}} - Y_{\alpha}|^2 \le C \times (1-\alpha)^2\right\} > 1 - \delta \tag{4}$$

where,

$$\begin{aligned} x^{\delta} &\equiv \inf\left\{x \in \mathcal{R}^n \mid P\left\{|X| > |x|\right\} < \delta\right\} \\ C &= \sigma_{max}(W_2 W_1)^2 |x^{\delta}|^2 + |W_2 b_1|^2 \end{aligned} \tag{5}$$

where, $\sigma_{max}(W_2 W_1)$ is the largest singular value of $W_2 W_1$ By choosing δ arbitrarily small, probability of error is bounded proportional to $(1-\alpha)^2$ with probability one. We capitalize on this by incorporating $(1-\alpha)^2$ into our loss function. This loss term serves to align Y_{α} with Y_{linear}, compressing the model into a shallower model as demonstrated in Figure 2. In simpler terms, as α approaches one, substituting Y_{linear} with Y_{α} incurs a bounded error probability proportional to $(1-\alpha)^2$.

In practical applications, **Dropout** (Srivastava et al., 2014) and **BatchNormalization** (Ioffe & Szegedy, 2015) are frequently used between the layers of a network. Dropout works by temporarily deactivating random neurons during training, which helps spread the learning across different neurons to avoid over-reliance on any particular subset of them. It is important to note that Dropout is not active during model deployment, meaning it behaves as if it is not there. This characteristic means that Dropout doesn't affect the layer collapsing process and can be left out of consideration.

BatchNormalization is a process described by the formula:

$$Y = \frac{X - E[X]}{\sqrt{Var[X]}} \times \gamma + \beta \tag{6}$$

where, X and Y are the input and output random variables. γ and β are learnable parameters, and $E[X]$ and $Var[X]$ are the mean and variance of the current batch. Algorithm 1 outlines the specific steps to be taken for adjusting weights and biases when BatchNormalization is present.

To measure how well our compression method works, we use the concept of "compression gain". This is calculated as the percentage reduction in the number of parameters of a layer, excluding bias and BatchNormalization parameters. The formula for compression gain is:

$$Gain = 1 - \frac{n_{in} \times n_{out}}{n_{hidden} \times (n_{in} + n_{out})} \tag{7}$$

where, n_{in}, n_{hidden}, and n_{out} denote the input size, hidden size, and output size of a 2-layer MLP, respectively.

An interesting aspect of this formula is that the gain can sometimes be negative. Positive gain happens when n_{hidden} is larger than the product of n_{in} and n_{out} divided by their sum. This means that MLPs designed with a "bottleneck" structure (where n_{hidden} is relatively small) may not see much benefit from our layer collapsing method. On the other hand, most modern MLP designs, which tend to expand the layer size ("widening" structure), are likely to experience significant positive gains from this method.

3.1. Adaptation for Convolutional Layers

Convolutional layers play a pivotal role in numerous neural network architectures. Essentially, convolution can be formulated as sparse fully connected layers with some redundancies. Their nature as linear transforms permits representation via circulant matrices. A circulant matrix is a matrix where the row vectors have the same elements and each row vector is rotated one element to the right relative to the preceding row vector. An interesting property of these matrices is that their product also forms a circulant matrix, resulting in another convolution layer. The kernel size of the resulting convolution layer, denoted as k, is calculated by $k = k_1 + k_2 - 1$ where, k_1 and k_2 are the kernel sizes of the original convolution layers. The compression gain can be calculated by:

$$Gain_{conv} = 1 - \frac{k_1^2 \times c_{in} \times c_{hidden} + k_2^2 \times c_{hidden} \times c_{out}}{(k_1 + k_2 - 1)^2 \times c_{in} \times c_{out}} \tag{8}$$

where, c_{in}, c_{hidden}, and k_1 correspond to the input channels, output channels, and the kernel size of the initial convolution layer, respectively. Meanwhile, c_{hidden}, c_{out} and k_2 are the parameters of the subsequent convolution layer. However, a crucial point to consider when applying LayerCollapse to common CNN architectures, such as ResNet, is that the gain often turns out to be negative. To achieve a positive gain, it is imperative for c_{hidden} to be significantly larger than both c_{in} and c_{out}. This observation highlights the need for careful consideration when attempting to collapse layers in such architectures.

3.2. Prior Distribution Analysis of Regularization

We extend the use of the parametric ReLU (PReLU) function to develop a regularization method based on the maximum a posteriori (MAP) framework (DeGroot, 2005). Aligning with Occam's razor (Schaffer, 2015), which favors simpler explanations, we propose to adopt a normal distribution with a mean of 1 as the prior distribution for α. This setting effectively limits the two layers to a linear space, reducing their functional complexity, as illustrated in Figure 1. This choice is backed by empirical results given in section 4. MAP framework suggests adding negative log likelihood of the prior distribution of parameters to the loss function. This is referred to as the prior loss or regularization loss. Following this principal the added regularization term is:

$$\mathcal{L}_{\text{reg}} = lc \times (1 - \alpha)^2, \tag{9}$$

where lc is the regularization constant. Notice this formulation matches perfectly to our intuition from equation (4).

Algorithm 1 Layer Collapse

```
 1: function COLLAPSE($W_1, b_1, W_2, b_2, \gamma, \beta, \bar{\mu}, \bar{\sigma}, \alpha, \tau$)
 2:     if $\alpha \leq \tau$ then
 3:         $s_{BN} \leftarrow \frac{\gamma}{\sqrt{\bar{\sigma}}}$
 4:         $W_{\text{new}} \leftarrow s_{BN} \times W_2 \cdot W_1$
 5:         $b_{\text{new}} \leftarrow W_2 \cdot (s_{BN} \times (b_1 - \bar{\mu}) + \beta) + b_2$
 6:         return $W_{\text{new}}, b_{\text{new}}$
 7:     else
 8:         return un-collapsible
 9:     end if
10: end function
```

4. Experiments

This section assesses the performance of LayerCollapse on benchmarks in both vision and NLP domains. We focus on its dual role: as an effective regularizer and in enabling post-training compression of neural networks.

We detail experiments in Section 4.2, where models are finetuned with LayerCollapse regularization and are subsequently compressed. Experiments in section 4.1 illustrate LayerCollapse 's capability not only as a compression tool for pretrained networks but also in enhancing model generalization when training, as evidenced on the Wikitext-103 language modeling benchmark (Merity et al., 2016). Additionally, we compare LayerCollapse against a modified version of the *LayerDrop* structured pruning technique, highlighting the unique advantages of our approach.

The final part of our analysis pits LayerCollapse against Knowledge Distillation (Hinton et al., 2015), with a focus

Table 2. Training performance on Wikitext-103 dataset. For base variants the modified layer is the last layer i.e. layer 12. In the large variant layers 11, 16, and 32 are selected for compression.

Model	Layers	Params (M)	PPL
GPT-2 (Radford et al., 2019)	0	124.4	**18.87**
GPT-2 + LayerDrop (Fan et al., 2019)	1	119.9(-4%)	19.29
GPT-2 + LayerCollapse (ours)	1	120.3(-3%)	19.09
GPT-2 Large (Radford et al., 2019)	0	774.0	17.65
GPT-2 Large + LayerDrop (Fan et al., 2019)	3	734.7(-5%)	15.61
GPT-2 Large + LayerCollapse (ours)	3	739.6(-4%)	**15.17**

on the computational resources required for each, despite targeting the same network architecture.

4.1. Regularization Performance of LayerCollapse

We employ LayerCollapse on the Open-AI's gpt2 model (Radford et al., 2019). We train this model on Wikitext-103 dataset (Merity et al., 2016) and evaluate on the test perplexity. We compare this method to an adaptation of *LayerDrop* (Fan et al., 2019) structured pruning in both regularization and zero shot compression performance.

Setup. In our experiments, we utilize the base and large versions of the GPT-2 causal language model (Radford et al., 2019), with a fixed context size of 128 tokens. Our layer selection methodology, inspired by (Fan et al., 2019), involves initially collapsing each layer in a one-shot manner, followed by an evaluation to identify layers most tolerant to compression. We conduct training for $3k$ steps, maintaining a batch size of 128 across all experiments. The learning rate scheduling is managed by LiveTune (Shabgahi et al., 2023), starting from an initial rate of 0.001. Optimization is performed using a modified version of the Adam optimizer (Kingma & Ba, 2014), with comprehensive details about this optimizer available in Appendix B. Exhaustive list of experiments details can be found in Appendix D. The comparison is with a modified version of *LayerDrop* (Fan et al., 2019). This modification, drawing inspiration from (Savarese & Figueiredo, 2017), employs gates for selected layers. These gates use a sigmoid function to dynamically alternate between activation and deactivation, enhancing training stability.

Results. As shown in Table 2, LayerCollapse surpasses *LayerDrop* in terms of final perplexity score. Notably, the large variant of GPT-2 tends to overfit, resulting in higher Perplexity compared to both regularized approaches. However, *LayerDrop* achieves greater compression gain.

Further experiments on the regularization performance of LayerCollapse is given in appendix C

4.2. LayerCollapse for Compression

We apply LayerCollapse to compress targeted layers in pre-trained models in both computer vision and sentimental analysis tasks. The process involves a few training steps with the regularization loss defined in Equation 9, tailored to the specific dataset and model in use. In all cases, we set the slope threshold for layer collapse at 0.0001. It's important to note that we report results directly after layer collapse, without any additional finetuning. For architectures such as Bert, ViT, and MLP-Mixer, we substitute GeLU activations in the compressed layers with ReLU. This change has been empirically found to have a minimal effect on model performance. Our compression approach focuses on the model's final layers, driven by the insight that these layers handle more abstract data representations, rendering them more suitable for compression. The compression is carried out sequentially, one layer at a time. For more information on the experimental framework, please refer to Appendix D.

4.2.1. Performance on GLUE

Setup. We assess LayerCollapse 's compression effectiveness on pre-trained Bert-Large and Bert-Base (Devlin et al., 2018) using the GLUE benchmark (Wang et al., 2018). We employ SGD with momentum as our optimization technique, setting the learning rate at 0.001. The *Average* column is slightly different than GLUE score because since we exclude WNLI as in (Devlin et al., 2018).

Results. As illustrated in table 3 LayerCollapse yields 15% compression gain for Bert base with about 1% score reduction on the GLUE benchmark. As evident the larger variant of Bert is prone to overfit thus LayerCollapse regularization improves the score in many tasks. As illustrated LayerCollapse provided better scores than provided in (Devlin et al., 2018). This is while the resulting model is 8% smaller in terms of parameter count.

4.2.2. Performance on ImageNet-1K

Setup. We apply LayerCollapse across a diverse set of vision models, including CNNs, MLP-Mixers (Tolstikhin et al., 2021), and Vision Transformers (Dosovitskiy et al.,

Table 3. Post-training compression GLUE test results illustrating accuracy and achieved compression through LayerCollapse across different layers in main variants of BERT. For QQP, MRPC average of F1 and accuracy are reported, for STS-B average of Spearman and Pearson is reported.

Model	Layers	Params (M)	MNLI(m/mm)	SST-2	STS-B	RTE	QNLI	QQP	MRPC	CoLA	Average
Bert Base	0	109.1	83.5/83.7	92.7	86.4	65.7	91.0	89.4	85.8	61.0	81.9
	4	92.5(-15%)	82.6/82.8	91.2	87.3	61.4	89.8	86.3	88.4	59.6	80.8
Bert Large	0	334.9	86.3/**86.1**	**93.5**	87.7	66.4	89.3	**89.9**	85.8	62.6	82.7
	4	309.7(-8%)	**86.6**/85.5	93.0	**89.5**	**71.5**	**91.4**	89.4	**87.5**	**62.6**	**83.8**

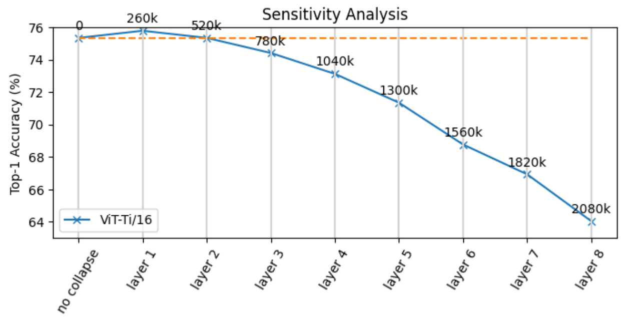

Figure 3. Layer-wise collapse accuracy and parameter reduction analysis for ViT-T/16.

2020), utilizing the ImageNet-1K benchmark (Russakovsky et al., 2015) for evaluation. Pretrained models are sourced from the timm library (Wightman, 2019). Optimization is performed using SGD with momentum, with a learning rate set to $5e^{-4}$. We employ the *Self Distillation* method (Fan et al., 2019). The loss function is comprised of three parts, cross entropy between labels and result, KL divergence between the output and original model output as the teacher and the LayerCollapse regularization term given in equation (9) with $lc = 0.2$. This process is capped at 20 epochs. Number of epochs for VGG architectures are given in Table 5. The MAC operations are all calculated by propagating a random 224×224 image through the model. Further details are enumerated in Appendix D.

Sensitivity Analysis. To quantify the robustness of LayerCollapse, we conduct a layer-by-layer sensitivity analysis on the Vision Transformer T/16 model, as shown in Figure 3. This removal of layers provides insight into the impact of incremental compression on the model's accuracy. Interestingly, we observe an initial increase in accuracy upon the collapse of the first layer, attributed to an improvement in generalization compared to the original model.

Results. Table 4 illustrates that LayerCollapse achieves a substantial reduction in model size-up to 74% in VGG11—with minimal accuracy degradation. Notably, models with heavier classification heads, such as VGG, can benefit significantly from our method. In ViT T/16, S/16, and B/16 variants we get roughly 5 percent reduction in model size per LayerCollapse. In all ViT variants we get a consistent reduction of over 10% in size and complexity with less than a 1% accuracy loss. In the Mixer architecture, 2 types of MLP layers known as MLP-token and MLP Channels exist. MLP-Channel has a significantly bigger size. We found that MLP-tokens can be collapsed in about an epoch of computation whilst MLP-channels require more effort. In the Mixer experiment, we collapsed the layers in pairs to ensure MLP-Channels are affected. In the Mixer architecture we get 14% reduction in size with less than a 1% accuracy degradation.

4.3. LayerCollapse vs Knowledge Distillation

Setup. Our evaluation framework utilizes batch-normalized VGG variants from the torchvision repository (maintainers & contributors, 2016), on ImageNet-1K (Russakovsky et al., 2015) providing a substantial dataset for training and validation. These variants are chosen for their prevalent use in compression studies and their capability to learn without prior training. To facilitate a fair comparison between LayerCollapse and Knowledge Distillation, we adopt the same target architecture for both methods. The rest of the training is set up identical to section 4.2 for both LayerCollapse and Knowledge distillation.

Comparison. The models are rigorously compared across several dimensions: training cost, the number of training epochs required, and the accuracy metrics—top-1 and top-5—on the validation set. The computational costs are quantified on a Nvidia RTX A6000 GPU.

Results. Referencing table 5, LayerCollapse consistently achieves over 10% higher accuracy while incurring only 30% of the computational cost compared to Knowledge Distillation. This trend is more pronounced in larger models, which exhibit greater compression potential. Showing up to 15% accuracy gain with 50% of tuning cost. It is noteworthy that for models necessitating pre-training, such as MLP-Mixer and ViT, the absence of pre-training data and the high computational demands often make Knowledge Distillation impractical. LayerCollapse emerges as a unique method for model compression through architectural modification, offering significant cost advantages. However, we recognize the inherent limitations of LayerCollapse in terms of the

Table 4. Post-training compression results, showcasing accuracy and compression attained via LayerCollapse across different layers in relevant architectures.

Model	Layers	Params (M)	MACs (G)	Top1 Acc. (%)
ViT T/16	0	5.72	1.08	75.35
	3	4.94(-14%)	0.91(-16%)	74.42
ViT S/16	0	22.05	4.24	81.38
	2	19.98(-9%)	3.84(-9%)	79.74
ViT B/16	0	86.57	16.85	81.43
	2	78.30(-10%)	15.23(-10%)	80.31
ViT L/16	0	304.33	59.66	84.80
	2	289.64(-5%)	56.77(-5%)	84.22
Mixer B/16	0	59.88	12.61	76.47
	4	51.39(-14%)	10.81(-14%)	75.67
VGG19	0	143.67	19.65	74.21
	2	45.11(-67%)	19.55(-1%)	71.22
VGG11	0	132.86	7.62	70.37
	2	34.31(-74%)	7.52(-1%)	66.36

Model	Method	Top1 Acc(%)	No. of Parameters (M)		Cost	Epochs
			Initial	Final		
vgg19	LC	71.22	143.67	45.11	11h	5
	KD	63.42			51h	23
vgg16	LC	70.01	138.36	39.80	14h	7
	KD	59.63			31h	16
vgg13	LC	67.21	133.05	34.49	16h	10
	KD	58.26			45h	28
vgg11	LC	66.36	132.86	34.31	20h	20
	KD	58.64			42h	42

Table 5. An analysis of LayerCollapse (LC) and Knowledge Distillation (KD) methods applied to VGG variants.

diversity of achievable target model architectures.

5. Conclusion

In this paper, we presented LayerCollapse , an innovative regularization-based method for neural network compression. LayerCollapse effectively merges linear layer pairs to decrease network depth and parameter count, resulting in reduced operational demands. A novel regularizer specifically designed for compression promotes the formation of shallower network structures.

Through comprehensive evaluations, LayerCollapse has shown promising regularization capabilities, often surpassing existing methods. In comparative studies with Knowledge Distillation, LayerCollapse not only exhibited enhanced computational efficiency but also consistently outperformed in terms of results, achieving over 10% improvement in performance while requiring 80% fewer resources. However, it is important to note that LayerCollapse 's advantages are predominantly observed in architectures utilizing MLPs with widening designs, and it does not significantly enhance state-of-the-art CNN architectures. Despite these limitations, LayerCollapse represents a notable advancement in the development of efficient neural network models, bridging the gap between high performance and manageable model size.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agirre, E., M'arquez, L., and Wicentowski, R. (eds.). *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007).* Association for Computational Linguistics, Prague, Czech Republic, June 2007.

Alvarez, J. M. and Salzmann, M. Learning the number of neurons in deep networks. *Advances in neural information processing systems*, 29, 2016.

Bar Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. The second PASCAL recognising textual entailment challenge. 2006.

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., and Magnini, B. The fifth PASCAL recognizing textual entailment challenge. 2009.

Cho, M., Joshi, A., Reagen, B., Garg, S., and Hegde, C. Selective network linearization for efficient private inference. In *International Conference on Machine Learning*, pp. 3947–3961. PMLR, 2022.

Dagan, I., Glickman, O., and Magnini, B. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pp. 177–190. Springer, 2006.

DeGroot, M. H. *Optimal statistical decisions*. John Wiley & Sons, 2005.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

do Santos, C. F. G., Colombo, D., Roder, M., and Papa, J. P. Maxdropout: deep neural network regularization based on maximum output values. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 2671–2676. IEEE, 2021.

Dolan, B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Fu, Y., Yang, H., Yuan, J., Li, M., Wan, C., Krishnamoorthi, R., Chandra, V., and Lin, Y. Depthshrinker: a new compression paradigm towards boosting real-hardware efficiency of compact neural networks. In *International Conference on Machine Learning*, pp. 6849–6862. PMLR, 2022.

Ghiasi, G., Lin, T.-Y., and Le, Q. V. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*, 31, 2018.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pp. 1–9. Association for Computational Linguistics, 2007.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Gupta, M. and Agrawal, P. Compression of deep learning models for text: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(4):1–55, 2022.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Haykin, S. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

Jha, N. K., Ghodsi, Z., Garg, S., and Reagen, B. Deepreduce: Relu reduction for fast private inference. In *International Conference on Machine Learning*, pp. 4839–4849. PMLR, 2021.

Jiang, N.-F., Zhao, X., Zhao, C.-Y., An, Y.-Q., Tang, M., and Wang, J.-Q. Pruning-aware sparse regularization for network pruning. *Machine Intelligence Research*, 20(1): 109–120, 2023.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Le, Y. and Yang, X. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Lemaire, C., Achkar, A., and Jodoin, P.-M. Structured pruning of neural networks with budget-aware regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9108–9116, 2019.

Levesque, H. J., Davis, E., and Morgenstern, L. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, pp. 47, 2011.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Liu, C. and Wu, H. Channel pruning based on mean gradient for accelerating convolutional neural networks. *Signal Processing*, 156:84–91, 2019.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

maintainers, T. and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Nguyen, H. D., Alexandridis, A., and Mouchtaris, A. Quantization aware training with absolute-cosine regularization for automatic speech recognition. 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pp. 2383–2392. Association for Computational Linguistics, 2016.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Savarese, P. and Figueiredo, D. Residual gates: A simple mechanism for improved network optimization. In *Proc. Int. Conf. Learn. Representations*, 2017.

Schaffer, J. What not to multiply without necessity. *Australasian Journal of Philosophy*, 93(4):644–664, 2015.

Shabgahi, S. Z., Sheybani, N., Tabrizi, A., and Koushanfar, F. Livetune: Dynamic parameter tuning for training deep neural networks. *arXiv preprint arXiv:2311.17279*, 2023.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pp. 1631–1642, 2013.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Sundaram, S. S., Khodke, Y., Li, Y., Jang, S.-J., Lee, S.-S., and Kang, M. Freflex: A high-performance processor for convolution and attention computations via sparsity-adaptive dynamic frequency boosting. *IEEE Journal of Solid-State Circuits*, 2023.

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *arXiv preprint 1805.12471*, 2018.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

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

Yang, Z., Cui, Y., Yao, X., and Wang, S. Gradient-based intra-attention pruning on pre-trained language models. *arXiv preprint arXiv:2212.07634*, 2022.

A. Proof of Equation (4)

Given $Y_{linear} = W_2W_1X + W_2b_1 + b_2$ and $Y_\alpha = W_2PReLU_\alpha(W_1X + b_1) + b_2$ we show that $\forall\delta > 0$

$$P\left\{|Y_{\text{linear}} - Y_\alpha|^2 \leq C \times (1-\alpha)^2\right\} > 1 - \delta \tag{10}$$

where,

$$\begin{aligned} x^\delta &\equiv \inf\left\{x \in \mathcal{R}^n \mid P\left\{|X| > |x|\right\} < \delta\right\} \\ C &= \sigma_{max}(W_2W_1)^2|x^\delta|^2 + |W_2b_1|^2 \end{aligned} \tag{11}$$

where, $\sigma_{max}(W_2W_1)$ is the largest singular value of W_2W_1.

We define indicator random variable I as:

$$I \equiv \begin{cases} 1 & W_1X + b_1 \geq 0 \\ 0 & otherwise \end{cases} \tag{12}$$

Writing Y_α using (12) we have:

$$Y_\alpha = IY_{linear} + (1-I)(\alpha W_2W_1X + \alpha W_2b_1 + b_2) \tag{13}$$

Using the definition of Y_{linear}, the squared distance of Y_{linear} and Y_α can be written as:

$$|Y_{\text{linear}} - Y_\alpha|^2 = (1-I)^2(1-\alpha)^2(W_2W_1X + W_2b_1)^2 \tag{14}$$

Since $(1-I)^2$ is either 0 or 1 we have:

$$|Y_{\text{linear}} - Y_\alpha|^2 \leq (1-\alpha)^2(W_2W_1X + W_2b_1)^2 \tag{15}$$

Using the triangle inequality we can write:

$$|Y_{\text{linear}} - Y_\alpha|^2 \leq (1-\alpha)^2(|W_2W_1X|^2 + |W_2b_1|^2) \tag{16}$$

It is known that:

$$|W_2W_1X|^2 \leq \sigma(W_2W_1)^2|X|^2, \tag{17}$$

where, $\sigma(W_2W_1)$ is the largest singular value of the W_2W_1 matrix. Using (17) we have:

$$|Y_{\text{linear}} - Y_\alpha|^2 \leq (1-\alpha)^2(\sigma(W_2W_1)^2|X|^2 + |W_2b_1|^2) \tag{18}$$

Using x^δ and δ as defined in equation (11) we have:

$$\begin{aligned} &P\{(1-\alpha)^2(\sigma(W_2W_1)^2|X|^2 + |W_2b_1|^2) \leq \\ &\quad (1-\alpha)^2(\sigma(W_2W_1)^2|x^\delta|^2 + |W_2b_1|^2)\} > 1 - \delta \end{aligned} \tag{19}$$

Using equations (19) and (18) with using the C defined in (11) we have:

$$P\left\{|Y_{\text{linear}} - Y_\alpha|^2 \leq C \times (1-\alpha)^2\right\} > 1 - \delta \tag{20}$$

B. AdamLC

Inspired by AdamW (Loshchilov & Hutter, 2017) we decouple the adam optimizer with our regularization so the adaptive learning rate does not make the regularization unpredictable the algorithm is given in 2. In this algorithm lc is the regularization constant.

Algorithm 2 AdamLC Optimization Algorithm

```
1: **function** TRAIN($\alpha, \beta_1, \beta_2, \epsilon, \theta_0, lc$)
2:     Initialize time step $t \leftarrow 0$
3:     Initialize first moment vector $m_0 \leftarrow \mathbf{0}$
4:     Initialize second moment vector $v_0 \leftarrow \mathbf{0}$
5:     Initialize parameter vector $\theta \leftarrow \theta_0$
6:     **while** not converged **do**
7:         $t \leftarrow t + 1$
8:         $g_t \leftarrow$ gradient at $\theta$
9:         $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$
10:        $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$
11:        $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
12:        $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
13:        $\theta \leftarrow \theta - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$
14:        **if** $\theta$ is negative slope **then**
15:            $\theta \leftarrow \theta - 2 \cdot (\theta - 1) \cdot lc \cdot \alpha$
16:        **end if**
17:    **end while**
18:    **return** $\theta$
19: **end function**
```

C. Study of Regularization on VGG

We provide an in-depth analysis of how our regularization approach impacts both the generalization and compression abilities in training image classifiers from scratch.

Dataset. Our study utilizes the CIFAR100 (Han et al., 2015) and TinyImageNet (Le & Yang, 2015) datasets, which are mid-sized and rich in diversity. The CIFAR100 dataset includes 60,000 images across 100 classes, and TinyImageNet, a subset of the larger ImageNet, comprises 100,000 images across 200 classes.

Implementation. The experiments are conducted using the VGG architecture because of its effective learning capabilities without pre-training and competitive performance in benchmarks. For consistency, all experiments utilize the same setup and hyper parameters. Note that the goal of this study is not to show state of the art performance using our proposed regularization, but to show its effectiveness in reducing over-fitting and final model size. In all experiments, traditional regularizers, such as data augmentation (rotation, random crop, and flipping), dropout(do Santos et al., 2021), and batch normalization(Ioffe & Szegedy, 2015), have been used. This is to show the added contribution of 9.

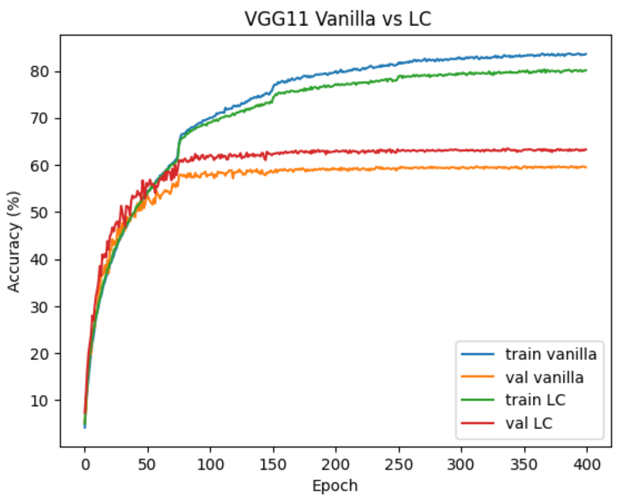

Figure 4. Evaluating the impact of LayerCollapse regularization on CIFAR100, we compare the top-1 accuracy of VGG11 both with and without this regularization technique.

Table 6. Number of parameters after Collapse and Top-1 Accuracy of various models with and without LC on CIFAR-100 and Tiny ImageNet.

Model	CIFAR-100		Tiny ImageNet	
	#Param	Top-1(%)	#Param	Top-1(%)
VGG11	11.7M	59.7	18.4M	60.2
VGG11(LC)	9.3M	63.5	9.6M	61.7
VGG13	11.9M	66.1	18.6M	50.3
VGG13(LC)	9.5M	67.8	9.8M	52.2
VGG16	17.2M	63.3	23.9M	48.9
VGG16(LC)	14.8M	64.1	15.1M	49.7
VGG19	22.5M	62.5	29.6M	46.8
VGG19(LC)	20.1M	63.8	20.4M	48.2

Training is done using Stochastic Gradient Descent (SGD) with momentum a learning rate scheduler, trained for a total of 400 epochs per experiment. The regularization is applied to 40% of the layers, starting with the latest layers, with the strength of 0.05 across all experiments. extensive experiment details can be found in Appendix D.4.

Discussion. The results, documented in Table 6, demonstrate the benefits of incorporating compression aware regularization. We observe an enhancement in top-1 accuracy by as much as 2% with 50% reduction in model size. The learning curve presented in Figure 4 suggests a narrowing gap between training and validation accuracy, indicating a reduction in overfitting and highlighting the effectiveness of our regularization strategy in improving generalization.

D. Experiment Details

The experimental files are accessible at [https://github.com/sohail2810/LayerCollapse].

D.1. Regularization Performance of LayerCollapse

Training was conducted using a modified version of the Adam optimizer, as detailed in Appendix B. A consistent regularization parameter, Regularization 5, was applied across all experiments, and the same value was used for comparisons with the *LayerDrop* method. The learning rate scheduler LiveTune (Shabgahi et al., 2023) was employed, with an initial learning rate set to 0.001.

In all experiments, a batch size of 128 was maintained, and the models were trained for a total of 3,000 steps. For performance evaluation, we applied a stride of 32 on the complete test dataset of Wikitext-103.

D.2. LayerCollapse for Compression

D.2.1. Performance on GLUE

Because of the diversity of tasks each have their own requirements for training (i.e. optimizer, learning rate, and epochs) All tasks except for QQP the SGD optimizer is used, and the modified adamLC B is used for QQP. In all experiments initial learning rate is 0.0001 which reduces to 0.00001 linearly. Collapsing is done one layer at a time. Starting from the final un-collapsed layer. In each step we reduce the regularization loss of the active layer during a retraining phase, when the loss is bellow $1e^{-4}$ we move to the next layer. Further details on the models used as a starte can be found in the code at [https://anonymous.4open.science/r/LayerCollapse-675B/README.md]. All tasks for Bert-base use batch size 128. Bert-large is trained on batch size 20.

D.2.2. Performance of ImageNet-1k

Dataset: We utilized the full ImageNet-1k dataset, with image dimensions of 224×224. Training images were augmented with random cropping, horizontal flipping, and rotation (up to 15 degrees). Validation images were not augmented. Normalization factors were model-specific: standard and mean values were set to 0.5 for all color channels in most models, except for the MLP-Mixer, which used the default ImageNet normalization (mean = $(0.482, 0.456, 0.406)$, std = $(0.229, 0.224, 0.225)$). The batch size was fixed at 256 for all experiments.

Self Distillation: We accelerated finetuning by using the original model's predicted labels, akin to Knowledge Distillation techniques. The loss function combined cross-entropy and KL Divergence terms:

$$loss = \frac{1}{2}CrossEntropy(p, \hat{p}) + \frac{1}{2}KLDivergance(p, \tilde{p}) + Regularization \tag{21}$$

where $\hat{p}$ and $\tilde{p}$ represent true and original model's predicted labels, respectively. Regularization is as defined in Eq. (9).

Layer collapsing was performed sequentially, starting from the last layer. Each layer's MLP was replaced with our CollapsibleMLP, featuring PReLU activations, and finetuned using Eq. (21) for up to 10 epochs. Following this, we applied the collapse procedure (Algorithm 1). A regularization strength of 0.05 and an SGD optimizer (momentum 0.9, initial learning rate 0.0005) were used. After 5 epochs, the learning rate was reduced to 10^{-4}.

D.3. LayerCollapse vs. Knowledge Distillation

For both LayerCollapse and Knowledge Distillation, the target architectures were identical. The LayerCollapse setup followed the procedure outlined in Section D.2.2. The distillation process employed the SGD optimizer with an initial learning rate of 0.001, reduced to 0.0001 upon plateau, and concluded when no further progress was possible.

D.4. Study of Regularization on VGG

To ensure comparability, all models, including those without LayerCollapse, shared the same training settings. The SGD optimizer with a starting learning rate of 0.005 was used, decreasing by a factor of 0.3 at epochs 75, 150, and 250. A dropout probability of 0.5 was applied. In LayerCollapse experiments, 0.4 fraction of layers were regularized with a strength of 0.05. Hyper-parameter tuning was not a focus of this study.