Layer Folding: Neural Network Depth Reduction using Activation Linearization

Amir Ben Dror*
amir.b@samsung.com
Niv Zehngut*
niv.z@samsung.com
Avraham Raviv*
avraham.r@partner.samsung.com
Evgeny Artyomov
evgeny.a@samsung.com
Ran Vitek
ran.vitek@samsung.com

Samsung Israel R&D Center
Tel Aviv
Israel

Abstract

Despite the increasing prevalence of deep neural networks, their applicability in resource-constrained devices is limited due to their computational load. While modern devices exhibit a high level of parallelism, real-time latency is still highly dependent on networks' depth. Although recent works show that below a certain depth, the width of shallower networks must grow exponentially, we presume that neural networks typically exceed this minimal depth to accelerate convergence and incrementally increase accuracy. This motivates us to transform pre-trained deep networks that already exploit such advantages into shallower forms. We propose a method that learns whether non-linear activations can be removed, allowing to fold consecutive linear layers into one. We use our method to provide more efficient alternatives to MobileNet and EfficientNet architectures on the ImageNet classification task. We release our code and trained models at https://github.com/LayerFolding/Layer-Folding.

1 Introduction

Multiple works have studied the relation between expressiveness and neural networks' depth. Early works [1, 6, 31, 41] showed that some deep neural networks cannot be represented by shallower networks unless those networks are exponentially wider. This exponential parameter growth of shallow networks compared to deeper ones representing the same function, has been widely studied [2, 5, 23, 33, 34, 37, 47]. While these findings suggest that a certain depth is required to preserve performance on a given task, many architectures are typically deeper than that. The role of the added layers can be viewed by the unrolled iterative estimation [9] – a group of successive layers iteratively refine their estimates of the

* Equal Contribution.

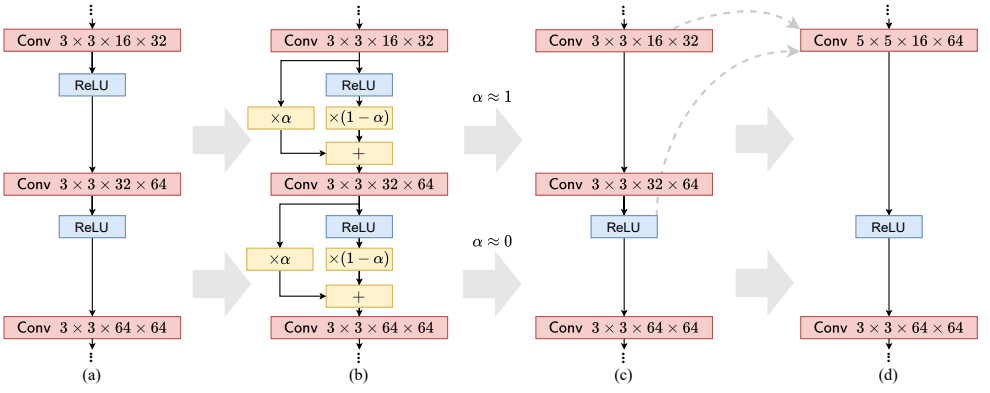

Figure 1: Illustration of our method. We replace activations of a given network $\mathcal{F}$ (a) with parametric activations according to Equation (1), resulting in $\mathcal{F}_\alpha$ (b). We fine-tune $\mathcal{F}_\alpha$ with the loss provided in Equation (4). When training converges, we remove activations whose $\alpha \approx 1$ (c). We fold consecutive linear operations, resulting in a shallower network $\mathcal{F}_{fold}$ (d). We then fine-tune $\mathcal{F}_{fold}$.

same features instead of computing entirely new representations. Hence, some layers in deep neural networks can be regarded as crucial for depth separation while others for refining representations and facilitating optimization, implying varying contribution to networks' expressiveness. Such observation plays an important role when efficiency is additionally considered as an optimization objective. We argue that some networks wield more layers than necessary, suggesting that their expressive power can be maintained with wider-but-shallower networks. This has been supported by recent works [17, 32] showing that deeper networks have a simplicity bias and encourage low-rank solutions. Shallower networks are particularly advantageous for hardware accelerators and GPUs that leverage intra-layer parallelism and suffer from inter-layer computational overhead.

We propose to learn which activations can be removed without incurring a significant accuracy degradation. This allows us to merge adjacent linear layers, and in turn, transform deep networks into shallow ones. Similarly to pruning methods [10, 11, 21] we focus on optimizing a pre-trained network. This allows the network to leverage the rich representations and local minimum obtained by the deeper network form and distill it to its shallow form. Recent pruning methods [42, 48] attenuate neurons during a fine-tuning phase, gradually reducing the network's size while allowing it to compensate. However, when applied to layer pruning, these methods force the network to gradually adopt new intermediate representations. The deep and shallow forms of a network may reside in local minima, for which traversing from one another may be challenging by gradient descent. In contrast, our optimization method maintains the intermediate representations of the original deep network during the fine-tuning phase. Recently, Ding et al. [7] suggested a similar decoupling between training-time architecture and inference-time architecture via structural re-parametrization to leverage parallel connections during training.

In line with other optimization methods, we focus on Convolutional Neural Networks (CNNs) for their prevalence in compute-intensive vision applications. We learn to remove non-linear activations between consecutive convolution layers, allowing their functionality-preserving merge. For layers with spatial kernels of size $k \times k$, this result in a larger $(2k -$

$1) \times (2k-1)$ kernel. We show how such a transformation, in spite of the added FLOPs, may reduce latency on different hardware devices. Interestingly, this comes in reverse of the common preference of multiple smaller kernels (e.g., 3×3) than fewer larger ones (e.g., 5×5) which originated in [39].

Merging multiple convolution layers may either increase or decrease the total number of FLOPs. It depends on the specific structure and width of those layers. Inverted bottleneck [38] is a commonly used building block for efficient networks. Merging layers of this block into a single convolution layer may cut FLOPs in half. We experiment on this architecture to show such potential gain. The fact that many recent works [14, 29, 40, 43, 45] have relied on neural architectures composed of inverted bottlenecks to achieve prominent performance over the ImageNet classification task [36] highlights the attractiveness of our method. We also show that our depth reduction method may be used jointly with channel pruning methods, improving efficiency even further.

Our contributions are as follows:

1. We propose *Layer Folding*, a novel method to reduce the depth of a neural network and fold consecutive linear layers by removing the non-linear activations that separate them. We show how our method facilitates optimization by maintaining the intermediate representations of the original depth.

2. We apply our method on efficient mobile networks over the ImageNet classification task [36] and improve their latency even further without a significant impact on their accuracy.

3. We perform extensive analysis over different network architectures and classification tasks and show how accuracy degrades as depth decreases. In particular, we show that networks' depth can be reduced with minimal accuracy drop.

2 Related Work

Pruning. Many pruning methods were proposed in order to improve a pre-trained network efficiency while maintaining its accuracy. Most pruning methods focus on pruning filters from each layer in a given network. Some methods allow pruning entire layers, in a way that resembles our focus on depth reduction. Chen and Zhao [3] proposed to estimate whether layers can be replaced with linear layers, sharing a similar motivation to ours. In contrast, they optimize layers independently while we train a network as a whole in an end-to-end manner, allowing layers to compensate for those which are removed. Wang et al. [44] proposed to remove blocks that have low discriminative power when compared to preceding blocks. As opposed to our method, the targeted blocks are abruptly removed from the network in a non-smooth manner. Neill et al. [30] proposed to remove layers based on layer similarity. They also empirically showed that there is a bound on the amount of compression that can be achieved before an exponential degradation in performance. We draw a different conclusion and show that such a bound originates from a network's depth rather than its size (i.e., number of parameters). Our work is in line with pruning methods that perform fine-tuning with additional loss which encourage more efficient networks, such as [46]. However, while such methods require the network to learn new intermediate representations, we maintain the rich representation space of the original network.

Neural Architecture Search (NAS). As high-capacity and high-performing neural networks became feasible to train, designing efficient architectures has gained high interest. Early works have suggested various design principles, such as adopting multiple smaller spatial kernels instead of larger ones [39], leveraging residual connections [13], decomposing weight matrices [15] and utilizing bottlenecks [38]. Current top-performing architectures are found using NAS. Since the search space spanned by possible architectures is intractable, existing methods use reinforcement learning [29], genetic algorithms [35], differentiable search [25] and other methods [19, 24, 27] to traverse it. Similarly to our method, differentiable methods learn architectural paths that allows the removal of entire layers, and even compensate it with added width to the preceding ones. However, NAS methods in general require costly computational resources for both training a super-network from scratch and covering multiple search space dimensions. Our method leverages a pre-trained network and focuses on a single search dimension: depth. Both allow an expedited training time and facilitated convergence.

Activation removal. Our method uniquely combines activation removal with consecutive linear layer folding. Nevertheless, activation removal alone has been used for other purposes. He et al. [12] proposed PReLU, generalizing the ReLU activation function. Ma et al. [28] extended this idea to arbitrary activation functions. These methods allow learning whether activations should be shifted towards identity. However, their method learns the extent of such shift with accuracy optimization in mind. In contrast, we learn a binary decision - keeping the original activation or replacing it with an identity - with efficiency optimization in mind. Activation removal has gained further attention with the resurgence of Private Inference (PI). PI performs inference on encrypted data, where latency is hindered mostly by non-linear activations such as ReLU. Ghodsi et al. [8] performed NAS to optimize the placement of skip connections and considered pruning ReLU activations that follow them. Jha et al. [18] proposed to measure ReLU criticality by evaluating a model's performance with ReLUs of entire stages or alternating layers being removed. While we leave it to future work, our work can also be used to accelerate PI.

3 Method

We present a method which allows to reduce the number of non-linear activations in a neural network. This effectively enables to merge adjacent linear layers into a single one. We call this process *Layer Folding*. Given a non-linear activation function σ, we define the parametric activation $\sigma_\alpha()$ to be the linear combination of σ and the identity function:

$$\sigma_\alpha(x) = \alpha x + (1-\alpha)\sigma(x), \quad 0 \leq \alpha \leq 1 \tag{1}$$

where α is a trainable parameter which provides an interpolation between σ and the identity function. When $\sigma = ReLU$, $\sigma_\alpha()$ is the common PReLU [12] activation. Given a trained neural network $\mathcal{F}$, we construct a network $\mathcal{F}_\alpha$ by transforming its activations into their corresponding parametric activations initialized with $\alpha = 0$. This ensures that $\mathcal{F}_\alpha$ maintains the same functionality of $\mathcal{F}$.

We perform a fine-tuning stage in which we optimize $\mathcal{F}_\alpha$ with respect to both the original task loss $\mathcal{L}_t$ and an auxiliary loss $\mathcal{L}_c$ that penalizes smaller α values, encouraging them to

become 1. We consider a general form of $\mathcal{L}_c$:

$$\mathcal{L}_c = \sum_{l \in L} c_l \, h(\alpha_l) \tag{2}$$

where α_l corresponds to the lth activation and $h(\alpha)$ is a monotonically decreasing function for $0 \leq \alpha \leq 1$. $\{c_l\}_{l \in L}$ weigh the contribution of each layer to $\mathcal{L}_c$. These are used to depict a varying potential value for folding different layers and can be set, for example, according to a measured latency on a target device. When we simply want to encourage a shallow network we set $c_l = 1$, $l = 1 : L$.

While many forms of h can be applied, we provide a simple suggestion for h such that $\mathcal{L}_c$ becomes:

$$\mathcal{L}_c = \sum_{l \in L} c_l \, (1 - \alpha_l^p) \tag{3}$$

We select this form for the following reasons: after training, a layer can be folded with its subsequent one only if its corresponding α is sufficiently close to 1. In particular, we are sensitive to small changes in α_l near 1, since the farther σ_α deviates from identity the larger the error incurred from Layer Folding. Yet, we are indifferent to small changes in α near 0 since the matching layer cannot be folded anyway. We would like $\mathcal{L}_c$ to represent these ideas. $p > 1$ is a hyperparameter controlling the flatness of the loss surface around $\alpha_l = 0$ and the strength in which larger α_l values are pushed to 1. Our final loss function is:

$$\mathcal{L} = \mathcal{L}_t + \lambda_c \, \mathcal{L}_c \tag{4}$$

where λ_c is a hyperparameter that balances between the task loss and the amount of layers that will be folded.

We define the folding of two adjacent linear layers that reside between the feature maps $\{X, Y, Z\}$, $g_1 : X \to Y$ and $g_2 : Y \to Z$, as their composite function, i.e., $g_1 \circ g_2 : X \to Z$. We provide two examples for fully connected layers and convolution layers while omitting the linear bias addition and batch normalization operations for brevity. For $g_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x}$, $g_2(\mathbf{y}) = \mathbf{W}_2\mathbf{y}$ fully connected layers where $\mathbf{x} \in \mathbb{R}^{d_X}$, $\mathbf{y} \in \mathbb{R}^{d_Y}$, $\mathbf{z} \in \mathbb{R}^{d_Z}$, $\mathbf{W}_1 \in \mathbb{R}^{d_Y \times d_X}$, $\mathbf{W}_2 \in \mathbb{R}^{d_Z \times d_Y}$, their folding is given by:

$$g_{fold}(\mathbf{x}) = \mathbf{W}_2\mathbf{W}_1\mathbf{x} \tag{5}$$

For $g_1(\mathbf{x}) = \left\{ \sum_{i=1}^{c_X} \mathbf{W}_1^{i,j} * \mathbf{x}_i \right\}_{j=1}^{c_Y}$, $g_2(\mathbf{y}) = \left\{ \sum_{j=1}^{c_Y} \mathbf{W}_2^{j,m} * \mathbf{y}_j \right\}_{m=1}^{c_Z}$ convolution layers where $\mathbf{x} \in \mathbb{R}^{h \times w \times c_X}$, $\mathbf{y} \in \mathbb{R}^{h \times w \times c_Y}$, $\mathbf{z} \in \mathbb{R}^{h \times w \times c_Z}$, $\mathbf{W}_1 \in \mathbb{R}^{k \times k \times c_Y \times c_X}$, $\mathbf{W}_2 \in \mathbb{R}^{k \times k \times c_Z \times c_Y}$, their folding is given by:

$$g_{fold}(x) = \left\{ \sum_{i=1}^{c_X} \left(\sum_{j=1}^{c_Y} W_1^{i,j} * W_2^{j,m} \right) * x_i \right\}_{m=1}^{c_Z} \tag{6}$$

Our method comprises 2 phases: *pre-folding* and *post-folding*. In the pre-folding phase we fine-tune $\mathcal{F}_\alpha$ with the loss defined in (4). When training converges we remove activations whose αs exceed a threshold τ and fold the corresponding adjacent layers, resulting in a shallower network $\mathcal{F}_{fold}$. In the post-folding phase, we fine-tune $\mathcal{F}_{fold}$ once more for two main reasons. First, the underlying function of $\mathcal{F}_{fold}$ may yet deviate from $\mathcal{F}_\alpha$ due to various

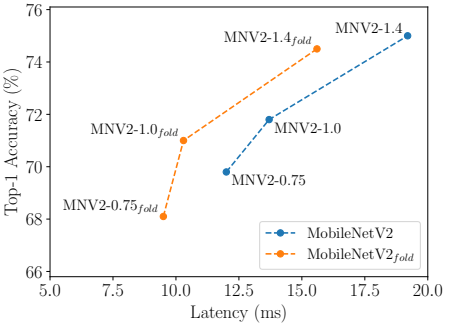

Figure 2: Layer Folding results on ImageNet.

Model	Acc. (%) / Acc. Drop (%)	Latency Reduction	FLOPs Reduction
MNV2-0.75	68.1 / 1.7	21%	4%
MNV2-1.0	71.0 / 0.8	25%	7%
MNV2-1.4	75.5 / 0.5	19%	3%
EffNet-lite0	74.6 / 0.5	15%	3%
EffNet-lite1	75.8 / 1.0	13%	0%

Table 1: Latency and FLOPs reduction obtained by applying Layer Folding on MobileNetV2 (MNV2) and EfficientNet (EffNet) on ImageNet.

layers' attributes such as padding, resulting in a small accuracy decrease. Post-folding fine-tuning allows the network to recover from it. Second, in some cases, $\mathcal{F}_{fold}$ result in a larger number of weights. Further training of $\mathcal{F}_{fold}$ may leverage the added capacity and increase accuracy. Our method is illustrated in Figure 1.

4 Depth Optimization for Efficient Networks

In this section we utilize our method to optimize networks with respect to both accuracy and efficiency. We perform our experiments on the ImageNet image classification task [36] and measure the latency of all models on NVIDIA Titan X Pascal GPU.

We consider the commonly used MobileNetV2 [38] and EfficientNet-lite [26]. We focus on these models for their attractiveness for hardware and edge devices, mostly credited to their competitive latency and the exclusion of squeeze-and-excite layers [16] employed by other state-of-the-art networks.

Both MobileNetV2 and EfficientNet-lite consist of multiple mobile inverted bottleneck blocks (MBConv). An MBConv block is composed of three convolution layers and two activations: (1) an expansion layer with $\mathbf{W} \in \mathbb{R}^{1\times1\times ct\times c}$ where t denotes an expansion factor over the input channel dimension c followed by ReLU6, (2) a depthwise layer with $\mathbf{W} \in \mathbb{R}^{3\times3\times c}$ followed by ReLU6, and (3) a projection layer with $\mathbf{W} \in \mathbb{R}^{1\times1\times c\times ct}$. Removing only the first ReLU6 will allow us to fold the expansion and depthwise layers into a convolution with $\mathbf{W} \in \mathbb{R}^{3\times3\times ct\times c}$. Interestingly, MobileNetEdgeTPU [14] followed this exact approach and adopted fused inverted bottleneck. In this model, expansion layers were folded with depthwise layers despite a FLOPs increase, acknowledging the potential latency reduction. This was performed on the first MBConvs. Indeed, in the general case, such folding may lead to a disadvantageous increase in FLOPs when $c \gg 9$. Removing the second ReLU6 will result in a similar FLOPs increase. However, we recognize that removing both ReLU6 activations will allow folding all three convolutions into a single layer with $\mathbf{W} \in \mathbb{R}^{3\times3\times c\times c}$. This effectively halve the computational load for blocks in MobileNetV2 where $t = 6$. In our experiments, we leverage this favorable scenario by forcing folding of an entire MBConv blocks. For every block, we share the same α among both of its activations. This ensures that they are either removed together or remain.

We apply Layer Folding on MobileNetV2-1.4, MobileNetV2-1.0, MobileNetV2-0.75, EfficientNet-lite0 and EfficientNet-lite1. Our implementation details are provided in the

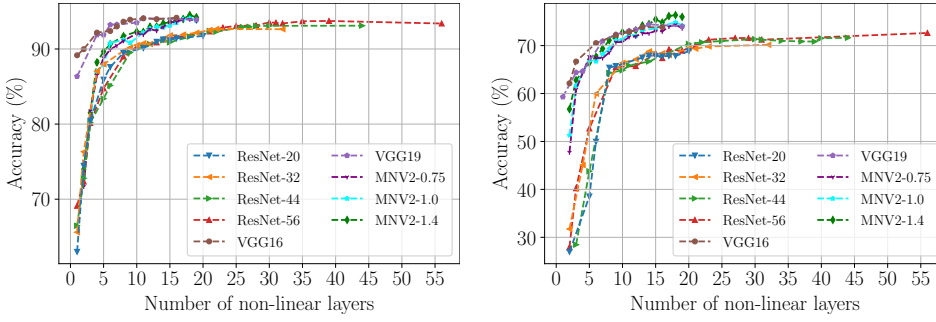

Figure 3: Layer Folding applied on ResNet, VGG, and MobileNetV2 (MNV2) architectures on CIFAR-10 (left) and CIFAR-100 (right). For each network, we gradually remove non-linear layers.

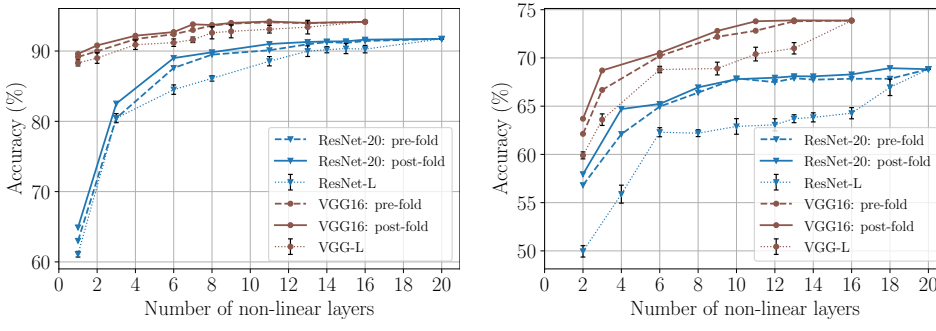

Figure 4: Relative contribution of pre-folding and post-folding. We perform pre-folding fine-tuning on ResNet-20 and VGG-16 and compare them to their shallower architectures resulting from post-folding. We also compare them to training the folded architectures from scratch, denoted by ResNet-L and VGG-L. Results are shown for CIFAR-10 (left) and CIFAR-100 (right).

Appendix.

As shown in Figure 2, our folded models outperform MobileNetV2 variants. For example, compared to MobileNetV2-0.75, we obtain a model with a 1.2% higher top-1 accuracy at a 14% faster execution time. Table 1 shows the latency reduction obtained for MobileNet and EfficientNet models. These improvements should be viewed while taking the simplicity and efficiency of our method into account.

5 Depth Reduction Analysis

In this section we use Layer Folding over different network architectures to evaluate how accuracy changes w.r.t. depth. We perform our experiments on CIFAR-10 and CIFAR-100 [20] image classification tasks. For CIFAR-10 and CIFAR-100, we consider the commonly used ResNet, VGG, and MobileNet architectures [13, 38, 39]. We use pre-trained ResNet models with depth $L \in \{20, 32, 44, 56\}$, VGG models with depth $L \in \{16, 19\}$, and MobileNetV2 with depth multiplier $d \in \{0.75, 1.0, 1.4\}$ [4]. For each of these networks, we apply Layer Folding with $c_l = 1$, $l = 1 : L$, $p = 2$, $\tau = 0.9$ while varying λ_c to obtain shallower networks

of varying depth. We apply only pre-folding in order to avoid increasing the networks' size.

Figure 3 shows our results for CIFAR-10 and CIFAR-100. Interestingly, all ResNet models exhibit a similar depth-accuracy tradeoff, regardless of their initial depth. For example, they all preserve an accuracy of 90% on CIFAR-10 when folded to only 10 non-linear layers. In addition, as depth decreases, they all depict a modest accuracy decrease at first and a noticeable drop afterwards, resembling a knee-point. For instance, the depth of ResNet-56 can be almost halved without incurring accuracy degradation. The accuracy of ResNet-32 folded to 6 non-linear layers, decreases by only 1% when folding another layer, but drops by more than 10% when folding two. Similar phenomena can be observed for VGG and MobileNet, as well as on CIFAR-100. Such analysis can aid network architecture design according to computational budget and required accuracy.

In Figure 4 we compare networks obtained by pre-folding and post-folding. The results show that the EDNL of a folded network conforms to the one of the original network. We note that for folded networks, shallower architectures utilize more parameters, as the folding of two consecutive convolution layers with weights $\mathbf{W} \in \mathbb{R}^{3\times3\times c\times c}$ result in a layer whose weights are $\mathbf{W} \in \mathbb{R}^{5\times5\times c\times c}$, i.e., its size grows by 40%. Hence, this experiment emphasizes the importance of depth even when it is disproportional to model's size, as networks were outperformed by deeper counterparts with fewer parameters. In addition, the slight accuracy increase of the folded networks quantifies the benefit of the post-folding phase. We believe that this makes them favorable to further efficiency optimization such as kernel size reduction. For example, we speculate that a 9×9 kernel resulted from folding four 3×3 kernels can be successfully distilled into a 7×7 kernel that nonetheless holds more capacity (49) than the four kernels altogether (36). We leave such optimization directions to future work.

We further compare our method to training randomly initialized networks with different depths as commonly practiced in NAS methods. Figure 4 shows the accuracy degradation of the folded architectures when trained from scratch rather than derived from their deeper source by our method.

While Layer Folding can be applied with various loss functions in accordance with Equation (2), we show that our chosen loss function in Equation (3) can effectively remove non-linear layers. Figure 5 validates that α values are indeed kept around zero or pushed to one, avoiding values in between. Additionally, we show that our method is not biased towards folding layers either in the beginning or the end of a given architecture. Table 2 shows the indices of the removed layers for folded ResNet, VGG and MobileNetV2 networks such that their resulting depth corresponds to the aforementioned knee-point in Figure 3.

6 Folding and Pruning

Layer Folding is a depth reduction method. As opposed to pruning methods, it does not encourage the network to explicitly remove any filters, weights, or neurons. This suggests that width reduction methods, such as filter pruning, can be used conjointly with our method, meaning that their expected contribution is additive rather than alternative to ours. In order to support this claim we applied the filter pruning method introduced by Li et al. [27] on ResNet-20 and ResNet-56 for both folded and non-folded networks. Table 3 shows that the accuracy drop in a folded network is similar or smaller than the accuracy drop in the original network.

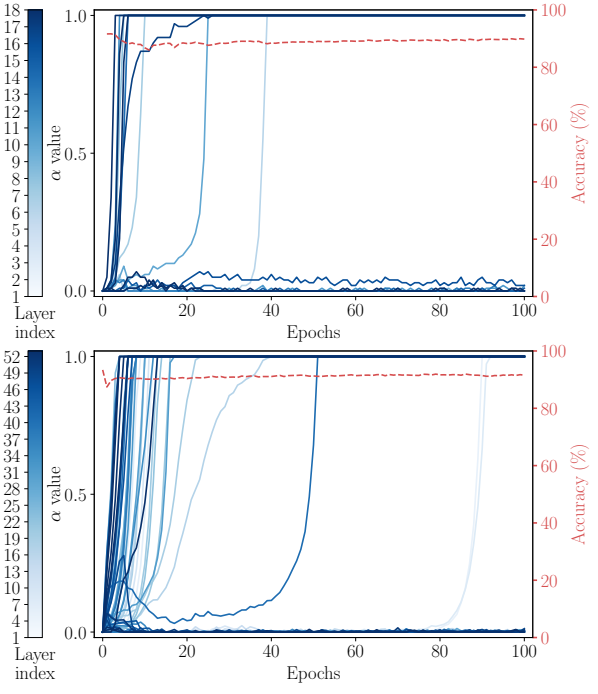

Figure 5: Progression of α_l values corresponding to non-linear layers in ResNet-20 (top) and ResNet-56 (bottom) throughout the pre-folding phase with $\lambda_c = 0.25$. As expected, all α values are either kept around zero or pushed to one.

Dataset	Model	Remaining activations	Depth	Acc. (%)
CIFAR-10	ResNet-20		9	89.82
	ResNet-32		9	90.02
	ResNet-44		9	89.88
	ResNet-56		10	90.29
	VGG16		9	93.89
	VGG19		8	93.23
	MNV2-0.75		10	91.90
	MNV2-1.0		10	92.03
	MNV2-1.4		9	92.31
CIFAR-100	ResNet-20		11	67.88
	ResNet-32		11	68.20
	ResNet-44		11	67.96
	ResNet-56		10	67.04
	VGG16		12	72.82
	VGG19		12	73.18
	MNV2-0.75		10	71.80
	MNV2-1.0		10	72.46
	MNV2-1.4		12	73.52

Table 2: Folded networks. Remaining activations are gray.

Model	Acc.	Acc. 20% Pruned	Acc. 40% Pruned
ResNet-20	91.27%	91.00% (-0.27%)	89.87% (-1.40%)
ResNet-20-10f*	90.33%	90.16% (-0.17%)	89.11% (-1.22%)
ResNet-56	94.37%	93.80% (-0.57%)	93.12% (-1.25%)
ResNet-56-37f†	92.69%	92.60% (-0.09%)	91.88% (-0.81%)

Table 3: Filter pruning results on CIFAR-10. *X%* Pruned denotes the rate of pruned filters in every inner layer. Pruning yields very similar accuracy drops for both folded and non-folded networks. (*) denotes ResNet-20 with 10 folded layers. (†) denotes ResNet-56 with 37 folded layers.

7 Conclusion

In this work we propose a novel method for removing non-linear activations. Extensive experiments on several image classification tasks show that our method can significantly reduce the depth of neural networks with a minor effect on accuracy. We show how reduced depth can aid latency reduction on hardware devices and provide efficient alternatives to mobile network architectures. We also show how Layer Folding can be used in conjunction with channel pruning methods, allowing for further efficiency improvements.

References

[1] Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565, 2014.

[2] Helmut Bolcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. Optimal approximation with sparsely connected deep neural networks. *SIAM Journal on Mathematics of Data Science*, 1(1):8–45, 2019.

[3] Shi Chen and Qi Zhao. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3048–3056, 2018.

[4] Yaofo Chen. Pytorch cifar models. https://github.com/chenyaofo/pytorch-cifar-models, commit:9751dd01a18d0c471b2c4522ae734757b6c94d8d, 2019.

[5] Amit Daniely. Depth separation for neural networks. In *Conference on Learning Theory*, pages 690–696. PMLR, 2017.

[6] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. *Advances in neural information processing systems*, 24:666–674, 2011.

[7] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. *arXiv preprint arXiv:2101.03697*, 2021.

[8] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. Cryptonas: Private inference on a relu budget. In H. Larochelle, M. Ranzato,

R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16961–16971. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/c519d47c329c79537fbb2b6f1c551ff0-Paper.pdf.

[9] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skn9Shcxe.

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1135–1143, 2015.

[11] Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

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

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[17] Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

[18] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. Deepreduce: Relu reduction for fast private inference. *CoRR*, abs/2103.01396, 2021. URL https://arxiv.org/abs/2103.01396.

[19] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal

transport. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2020–2029, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/f33ba15effa5c10e873bf3842afb46a6-Abstract.html.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[22] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

[23] Shiyu Liang and R. Srikant. Why deep neural networks for function approximation? In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SkpSlKIel.

[24] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.

[25] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[26] R Liu. Higher accuracy on vision models with efficientnet-lite. *TensorFlow Blog.[online] Available at: https://blog. tensorflow. org/2020/03/higher-accuracy-on-visionmodels-with-efficientnet-lite. html [Accessed 30 Apr. 2020]*, 2020.

[27] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *arXiv preprint arXiv:1808.07233*, 2018.

[28] Ningning Ma, Xiangyu Zhang, Ming Liu, and Jian Sun. Activate or not: Learning customized activation. *arXiv preprint arXiv:2009.04759*, 2020.

[29] Niv Nayman, Yonathan Aflalo, Asaf Noy, and Lihi Zelnik-Manor. Hardcore-nas: Hard constrained differentiable neural architecture search. *arXiv preprint arXiv:2102.11646*, 2021.

[30] James O' Neill, Greg Ver Steeg, and Aram Galstyan. Compressing deep neural networks via layer fusion. *arXiv preprint arXiv:2007.14917*, 2020.

[31] Razvan Pascanu, Guido Montúfar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. In *International Conference on Learning Representations (ICLR 2014)*, 2014.

[32] Guillermo Valle Pérez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *stat*, 1050:23, 2018.

[33] Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

[34] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[35] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[37] Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In *International Conference on Machine Learning*, pages 2979–2987. PMLR, 2017.

[38] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

[40] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497. Springer, 2019.

[41] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

[42] Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xCxXwTzx4L1.

[43] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020.

[44] Wenxiao Wang, Shuai Zhao, Minghao Chen, Jinming Hu, Deng Cai, and Haifeng Liu. DBP: discrimination based block-level pruning for deep model acceleration. *CoRR*, abs/1912.10178, 2019. URL http://arxiv.org/abs/1912.10178.

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

[46] Pengtao Xu, Jian Cao, Fanhua Shang, Wenyu Sun, and Pu Li. Layer pruning via fusible residual convolutional block for deep neural networks. *arXiv preprint arXiv:2011.14356*, 2020.

[47] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[48] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Advances in Neural Information Processing Systems*, 33, 2020.

[49] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.