DEEPRESHAPE: REDESIGNING NEURAL NETWORKS FOR PRIVATE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

The increased demand for privacy and security has given rise to private inference (PI), where inferences are made on encrypted data using cryptographic techniques. A challenge with deploying PI is computational and storage overheads, which makes them impractical. Unlike plaintext inference, PI's overheads stem from non-linear operations,i.e., ReLU. Despite the inverted neural operator overheads, all the previous ReLU-optimizations for PI still leverage classic networks optimized for plaintext. This paper investigates what PI-optimized network architectures should look like, and through thorough experimentation, we find that wider networks are more ReLU efficient and that how ReLUs are allocated between layers has a significant impact. The insights are compiled into a set of design principles (DeepReShape) and used to synthesize specific architectures (HybReNet) for efficient PI. We further develop a novel channel-wise ReLU dropping mechanism, ReLU-reuse, and achieve upto 3% accuracy boost. Compared to the state-of-the-art (SNL on CIFAR-100), we achieve a 2.35% accuracy gain at 180K ReLUs. For ResNet50 on TinyImageNet our method saves 4.2$\times$ ReLUs at iso-accuracy.

1 INTRODUCTION

The trend of processing machine learning inferences in the cloud has given rise to privacy concerns and inspired so-called private inference (PI). In PI, a client sends their (secured/encrypted) input data to the cloud service provider. The provider then performs the inference such that they do not learn anything about the client's data while the client learns nothing about the provider's trained network. PI is achieved via cryptographic primitives, which are effective but incur orders of magnitude higher processing latency than plaintext. In contrast to plaintext inference, most of the computational complexity of PI stems from non-linear layers, namely ReLU (Liu et al., 2017; Juvekar et al., 2018; Mishra et al., 2020; Rathee et al., 2020; Ghodsi et al., 2020; 2021; Cho et al., 2022a; Tan et al., 2021; Wang et al., 2022). To mitigate this, prior work has proposed ReLU-optimization techniques to prune the insignificant ReLUs for more efficient PI (Mishra et al., 2020; Lou et al., 2021; Jha et al., 2021; Cho et al., 2022b). However, as shown below, the merits of these ReLU optimization techniques are strongly dependent on the input baseline networks.

Performance of ReLU optimization is strongly correlated with the selection of baseline networks: Table 1 list the baseline networks used in recently proposed ReLU optimization papers, and Figure 1 and Figure 2 show the ReLU-accuracy tradeoffs for state-of-the-art coarse-grained (DeepReDuce (Jha et al., 2021)) and fine-grained (SNL (Cho et al., 2022b)), ReLU optimizations, with different baseline networks. In DeepReDuce, there exist substantial accuracy differences at iso-ReLU counts depending on which network is used—12.9% and 11.6% for networks with high and low ReLU operators. Even with fine-grained ReLU optimization (SNL), for networks pertained using the same family of networks there is a significant accuracy difference at iso-ReLU counts, which is more pronounced at lower ReLU counts. For example, while accuracy is similar at high ReLU counts (220K and 180K), there is a 3.15% and 4.6% accuracy difference at 25K and 15K ReLUs, respectively, between ResNet18 and WideResNet16x8. In fact, SNL achieves state-of-the-art results using two different baseline networks—WideResNet22x8 for higher ReLU counts and ResNet18 for lower ReLU counts.

Challenges for designing PI-friendly baseline networks: In addition, since the costs of PI are so different, we need new networks. Conventional network design (i.e., networks optimized for

ReLU optimization method	Baseline
Delphi (Mishra et al., 2020)	ResNet32
SAFENets (Lou et al., 2021)	ResNet32, VGG16
DeepReDuce (Jha et al., 2021)	ResNet18, ResNet9
SNL (Cho et al., 2022b)	ResNet18, WRN22x8

Table 1: Previously proposed ReLU optimization methods used different (input)baseline networks for the Pareto frontier of Accuracy-Latency in private inference.

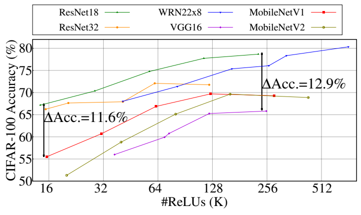


Figure 1: DeepReDuce with different baseline networks.

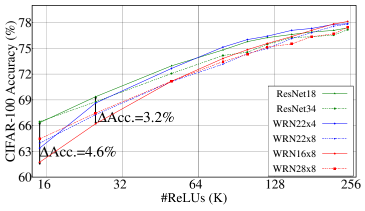


Figure 2: SNL with different baseline networks.

plaintext inference) seeks to maximize accuracy per FLOPs while ignoring ReLUs as the operators are effectively free. On th other hand, most prior work on optimizing PI uses networks optimized for plaintext inference. To the extent of our knowledge, there are two prior works on optimizing networks for private inference that rely on neural architecture search for designing PI-friendly networks. CryptoNAS (Ghodsi et al., 2020) and SPHYNX (Cho et al., 2022a) use the notion of ReLU-balancing and achieve ReLU-efficiency while disregarding their FLOPs implications, assuming FLOPs are free. However, a recent work (Garimella et al., 2022) shows that while ReLUs is the primary bottleneck, FLOPs are not free. Thus, a general design principle for FLOPs-ReLU-Accuracy balance is the need of the hour in PI. Nonetheless, a two-fold challenge exists in designing PI-friendly networks: (1) the lack of understanding of the correlation of ReLU and FLOPs counts with the network's design hyperparameters (e.g., width and depth of the network), and (2) the fundamental understanding for the desirable intrinsic properties (e.g., distribution of ReLUs) of baseline networks suitable for a wide range of ReLU counts; however, agnostic to the ReLU optimization techniques is largely missing.

To this end, we perform an exhaustive characterization to understand the interplay of ReLUs and FLOPs with network design hyper-parameters, summarized in four observations in Section 2.2. Based on these insights, we propose a novel design principle *ReLU equalization*, and devise a method *DeepReShape* to redesign the classical networks for ReLU equalization. Eventually, we demonstrate their benefits in conjunction with ReLU optimization methods. We achieve a 2.35% accuracy gain at 180K (on CIFAR-100) compared to the state-of-the-art fine-grained ReLU optimization SNL(Cho et al., 2022b), and a 4.2× ReLUs saving at iso-accuracy on ResNet50(TinyImageNet) compared to state-of-art coarse-grained ReLU optimization DeepReDuce (Jha et al., 2021).

Our Contributions:

1. We perform ReLU efficiency characterization — an unexplored facet and a crucial bottleneck for PI — over classical networks and discover their correlation with the network's depth and width. We find that width is much more important than depth as the network's complexity per unit of nonlinearity depends on the network's width; however, independent of the network's depth.
2. We propose *ReLU-equalization* — a novel design principle for redistributing the network's ReLU in their criticality order — and demonstrate its applicability also for designing the FLOPs-efficient networks such as RegNet (Radosavovic et al., 2020) and ConvNeXt (Liu et al., 2022b). Based on this, we devise a method *DeepReShape* for designing ReLU-efficient baseline networks, *HybReNet* (HRN), over a wide-range of ReLU counts with FLOPs-ReLU-Accuracy balance.
3. We reveal a "Capacity-Criticality Tradeoff" — a crucial factor for selecting the baseline networks for very low ReLU budget — for both coarse-grained and fine-grained ReLU optimizations. Furthermore, we propose *ReLU-reuse*, a channel-wise ReLU dropping technique to achieve very low ReLU counts, and gain accuracy boost upto 3% at iso-ReLU when employed with DeepReDuce.

2 Interplay of FLOPs and ReLUs with Design Hyperparameters

In this section, we start with defining the notations used in this paper. Further, we study the interplay of ReLUs and FLOPs with network design hyperparameters (primarily, width and depth).

2.1 Preliminary

Architectural building blocks: Figure 3 illustrates a schematic view of a standard four-stage network with building blocks and design hyperparameters. Similar to ResNet (He et al., 2016), network

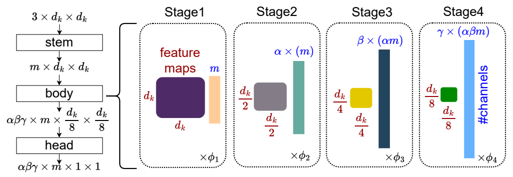

$3 \times d_k \times d_k$
stem
$m \times d_k \times d_k$
body
$\alpha\beta\gamma \times m \times \frac{d_k}{8} \times \frac{d_k}{8}$
head
$\alpha\beta\gamma \times m \times 1 \times 1$
Stage1
Stage2
Stage3
Stage4
feature maps
m
d_k
$\times\phi_1$
$\alpha \times (m)$
$\frac{d_k}{2}$
$\times\phi_2$
$\beta \times (\alpha m)$
$\frac{d_k}{4}$
$\times\phi_3$
$\gamma \times (\alpha\beta m)$
$\frac{d_k}{8}$
#channels
$\times\phi_4$

Figure 3: Depiction of architectural parameters and feature dimensions in a standard four stage network (width and depth hyperparameters are defined in the text below). For ResNet18 (on CIFAR and TinyImageNet), m = 64, ϕ_1=ϕ_2=ϕ_3 =ϕ_4=2, and α=β=γ=2.

	Stage1	Stage2	Stage3	Stage4
#Params	$\phi_1(m^2 f^2)$	$\phi_2(\alpha^2 m^2 f^2)$	$\phi_3(\alpha^2\beta^2 m^2 f^2)$	$\phi_4(\alpha^2\beta^2\gamma^2 m^2 f^2)$
#FLOPs	$\phi_1(m^2 d_k^2 f^2)$	$\phi_2(\frac{\alpha^2}{4} m^2 d_k^2 f^2)$	$\phi_3(\frac{\alpha^2\beta^2}{16} m^2 d_k^2 f^2)$	$\phi_4(\frac{\alpha^2\beta^2\gamma^2}{64} m^2 d_k^2 f^2)$
#ReLU	$\phi_1(m d_k^2)$	$\phi_2(\frac{\alpha}{4} m d_k^2)$	$\phi_3(\frac{\alpha\beta}{16} m d_k^2)$	$\phi_4(\frac{\alpha\beta\gamma}{64} m d_k^2)$
$\frac{\#Params}{\#ReLU}$	$m(\frac{f^2}{d_k^2})$	$\alpha m(\frac{4f^2}{d_k^2})$	$\alpha\beta m(\frac{16f^2}{d_k^2})$	$\alpha\beta\gamma m(\frac{64f^2}{d_k^2})$
$\frac{\#FLOPs}{\#ReLU}$	mf^2	$\alpha m f^2$	$\alpha\beta m f^2$	$\alpha\beta\gamma m f^2$

Table 2: Stagewise counts for the given width and depth hyperparameters ($f \times f$ is the spatial size of kernel, e.g., 3×3). We note that complexity per unit of nonlinearity ($\frac{\#Params}{\#ReLUs}$ and $\frac{\#FLOPs}{\#ReLUs}$) depends on the width; however, independent of the network's depth.

has a stem cell (that increases the channel count from 3 to m), followed by the network's main body (composed of linear and nonlinear layers, and performs most of the computation), followed by a head (usually a fully connected layer) that outputs the scores for the output classes. The network's main body is composed of a sequence of four stages, and the spatial dimensions of feature maps ($d_k \times d_k$) are progressively reduced by 2× (except the Stage1) in each stage, and feature dimensions remain constant within a stage. We keep the structure of stem cell and head fixed, and change the structure of the network's body using the design hyperparameters.

Definitions and design hyperparameters: Each stage is composed of identical blocks[1] repeated ϕ_1, ϕ_2, ϕ_3, and ϕ_4 times in Stage1, Stage2, Stage3, and Stage4 (respectively). These values are known as *stage compute ratios* and determine the network's overall depth. The output channels in stem cell (m) is known as *base channels*, and the number of channels progressively increases (except in Stage1) by a factor of α, β, and γ in Stage2, Stage3, and Stage4 (respectively). We call these values as *stagewise channel multiplication factors* (denoted as $ChMulFact$). The aforementioned depth and width hyperparameters primarily determine the distribution of FLOPs, ReLUs, and parameters in the network. When we widen the network: (1) by augmenting m, that increases the #channels in each layer by same factor, we denote this network as $BaseCh$ (e.g., from m=64 in ResNet18 to m=128); and (2) by (homogeneously) augmenting the $ChMulFact$s, we denote this network as $StageCh$ (e.g., from (α, β, γ) = (2, 2, 2) in ResNet18 to (α, β, γ) = (3, 3, 3)).

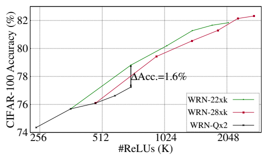
2.2 ReLU Efficiency: Depth vs Width

We now discuss our observations for ReLU efficiency and motivate the need for redesigning the classical networks for fast and efficient PI.

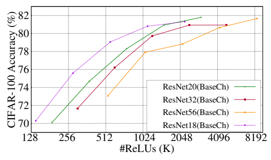
Observation 1: Wider networks are more ReLU-efficient than the deeper networks. For understanding the impact of network's depth and width on their ReLU-efficiency, we perform our experiments on WideResNet (Zagoruyko & Komodakis, 2016) — extensively used for demonstrating the benefits of width (Li et al., 2018; Park et al., 2019; Somepalli et al., 2022; Mirzadeh et al., 2022; Liu et al., 2022a)— and ResNet models. As shown in Figure 4(a), WRN-22xk outperforms WRN-28xk for all values of k, and the accuracy gap between WRN-Qx2 and WRN-22xk models increases with Q. Precisely, at an iso-ReLU count of 705K WRN22x4 outperforms WRN40x2 by **1.6%**. Similar observations hold true for ResNet models where the $BaseCh$ models of ResNet18 outperform the other deeper ResNets (Figure 4(b)). Thus, in contrast with the orthodox benefits of depth, it suggests that wider models have better ReLU-efficiency than their deeper counterparts.

To understand the superiority of wider networks for ReLU-efficiency, we investigate how the network's complexity, measured in terms of FLOPs and parameter counts (Radosavovic et al., 2019), grows when network's depth/width increases. In Table 2 we compute the stagewise parameters, FLOPs, ReLU counts and network's complexity per units of nonlinearity in terms of $\frac{\#Params}{\#ReLUs}$ and $\frac{\#FLOPs}{\#ReLUs}$. FLOPs and Parameters counts have a quadratic dependency on the channel width and vary linearly with the depth; whereas, ReLU count varies linearly with the channel width and depth, of their respective stages. *Interestingly,* the parameters and FLOPs per unit of ReLU is independent of depth hyperparameters ϕ_1, ϕ_2, ϕ_3, and ϕ_4; however, depends on the width of the network (m,

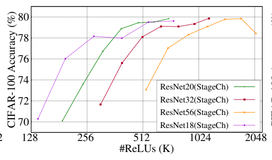
[1]Except the first block (in all but Stage1) which performs downsampling of feature maps by 2×.



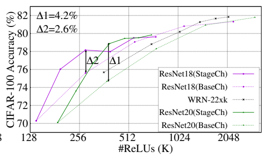
(a) WideResNet: Depth vs width



(b) ResNet: Base-channel variation



(c) ResNet: Stage-channel variation



(d) All width optimization together

Figure 4: ReLU-efficiency for depth vs width. (a) Width of WideResNet models WRN-22xk and WRN-28xk is increased by augmenting the $k \in \{2, 4, 6, 8, 10, 12\}$ and depth of WRN-Qx2 is increased by augmenting $Q \in \{16, 22, 28, 34, 40\}$; (b) width of $BaseCh$ models is increased by augmenting $m \in \{16, 32, 64, 128, 256\}$; (c) width of $StageCh$ models is increased by augmenting (α, β, γ) $\in \{(2, 2, 2); (3, 3, 3); (4, 4, 4); (5, 5, 5); (6, 6, 6); (7, 7, 7); (8, 8, 8)\}$; and (d) all width optimization together. *Wider networks exhibit better ReLU-efficiency than deeper networks.*

α, β, and γ). In other words, to achieve a similar level of complexity wider network requires fewer ReLUs. We note that our observation for the connection between stagewise complexity and width of the network resonates with the observation made in Radosavovic et al. (2019) and Dollár et al. (2021). The former showed that a model family with wider networks has a heavy-tail distribution of higher complexity networks, and the latter showed that the scaling width of the network results in the slowest growth in the number of activations.

Observation 2: Widening networks by homogeneously augmenting the stagewise channel multiplication factors substantially improves ReLU-efficiency. The benefit of width for higher ReLU efficiency, as evident from the above experimentation, raise a natural question: *to what extent a network can enjoy the benefit of increasing the width for higher ReLU-efficiency?* To answer this, we seek a more aggressive way of increasing the network's width and find that the growth of channels in the successive stages of almost all the classical networks (including WideResNets) is restricted by fixed values of (α, β, γ) = (2, 2, 2). To this end, we set α, β, and γ as design hyperparameters and increase their values homogeneously from (2, 2, 2) to (8, 8, 8). The experimental results are shown in Figure 4(c), and the accuracy saturation at higher ReLU is explained in Appendix D.2.

We compare ReLU-accuracy tradeoff for all the width optimization together in Figure 4(d). We make the following observations: (1) accuracy difference of $BaseCh$ and $StageCh$ networks grows until the saturation occurs in the latter, and $StageCh$ networks of ResNet18 and ResNet20 outperform their respective $BaseCh$ networks by **2.6%** and **4.2%** at $\sim 280K$ and $\sim 380K$ ReLU, respectively; and (2) At iso-ReLU count, $StageCh$ ResNet18 and ResNet20 networks outperform the WRN22x2 by ∼**2.3%** and ∼**3.2%** (respectively). Conclusively, ResNet18 $StageCh$ networks outperform all the networks before the saturation; hence, we select this as a baseline network for the rest of paper. Note that, compared to $BaseCh$, $StageCh$ widens the networks more aggressively; thus, further increasing the complexity per unit of ReLU and requiring even fewer ReLUs for a given complexity.

Observation 3: Neither the classical FLOPs-efficient networks nor the homogeneous stagewise channel scaling provides FLOPs-ReLU-Accuracy balance. Table 3 shows the change in layerwise trends of FLOPs and ReLUs, for a network with ϕ_1=ϕ_2=ϕ_3=ϕ_4 (e.g., ResNet18), when network's width is increased by (homogeneously)augmenting $ChMulFact$s. Notably, the conventional choice of (α, β, γ)=2 in classical networks makes FLOPs constant across the layers; however, in the network with $ChMulFact$s$>$ 2 the FLOPs count increases from initial layers to deeper layer (more rapidly at higher values of $ChMulFact$s). Nonetheless, in contrast with layerwise FLOPs, the layerwise ReLU trend is *distinct*. Precisely, for a network with $ChMulFact$s$<$ 4 ReLU count decreases from the initial layer to the deeper layer (more rapidly at lower values of $ChMulFact$s), and becomes constant for a network with $ChMulFact$s=4, and increases throughout the network at $ChMulFact$s$>$ 4. Recall that (Figure4(c)), before saturation, ReLU-efficiency of $StageCh$ networks increases with higher values of $ChMulFact$s; however, there is an exponential growth in FLOPs at higher values of $ChMulFact$. Since FLOPs are not free in PI (Garimella et al., 2022), a natural question is: how to establish FLOPs-ReLU-Accuracy balance? That is, *can a heterogeneous set of (α, β, γ) achieves ReLU-efficiency at par with $StageCh$ networks; however, with fewer FLOPs? If yes, how to find those heterogeneous sets?*

Observation 4: Width scaling by homogeneously augmenting the stagewise channels changes the distribution of ReLUs and reduces the proportion of non-critical ReLUs. We plot the stage-

Table 3: The (normalized) stagewise FLOPs and ReLUs, and layerwise (from initial to deeper layers) trends of FLOPs and ReLUs when network's width is increases by (homogeneously)augmenting α,β, and γ. *Distinct homogeneous* sets of (α, β, γ) are required for FLOPs and ReLU efficiency.

	Stage1	Stage2	Stage3	Stage4		(α,β,γ)=2	2<(α,β,γ)<4	(α,β,γ)=4	(α,β,γ)>4
FLOPs	1	$\frac{\alpha^2}{4}$	$\frac{\alpha^2\beta^2}{16}$	$\frac{\alpha^2\beta^2\gamma^2}{64}$	Layerwise FLOPs	constant	increasing (↑)	increasing (↑↑)	increasing (↑↑↑)
ReLUs	1	$\frac{\alpha}{4}$	$\frac{\alpha\beta}{16}$	$\frac{\alpha\beta\gamma}{64}$	Layerwise ReLUs	decreasing (↓↓)	decreasing (↓)	constant	increasing (↑)

wise ReLUs' distribution for ResNet18 $BaseCh$ and $StageCh$ networks in Figure 5(a) and Figure 5(b), respectively. Evidently, the distribution of ReLUs remains unchanged when network's width is scaled by augmenting m, as the #ReLUs in all the layers are scaled by the same degree. In contrast, the distribution of ReLUs changes in $StageCh$ networks, since augmenting $ChMulFact$s alters the stagewise ReLU count by different degrees (in particular, more aggressively in deeper layers). Consequently, as shown in Figure 5(b), the fraction of ReLUs in Stage1 decreases while that in the Stage4 increases with higher values of $ChMulFact$s. Also, initially at lower values of $ChMulFact$s fraction of ReLUs in Stage3 increases, and that in Stage2 remains constant. However, later at higher values of $ChMulFact$s, the fraction of ReLUs in Stage3 remains constant while that in Stage 2 decreases. Similar to Jha et al. (2021), we perform stagewise criticality analysis in Table 6 which shows the criticality order, from most to least critical, Stage3 $>$ Stage2 $>$ Stage 4 $>$ Stage1. Thus, conclusively, $StageCh$ networks have a reduced proportion of inconsequential ReLUs while the distribution of ReLUs in other stages does not follow their criticality order.

From above insights, we propose a novel design principle *ReLU equalization* (see Figure 10), and devise a method *DeepReShape* to redesign the classical network and achieves ReLU equalization. Based on this, we design a novel family of networks *HybReNet* (see Table 7).

3 DeepReShape

In this section, we begin by describing the DeepReShape method to achieve ReLU equalization. Then, we apply the same on a standard four-stage network and design *HybReNet* (HRN) networks.

ReLU-Equalization Algorithm: The DeepReShape method, described in Algorithm 1, is an iterative process where in each iteration the network design (primarily the width/depth) hyperparameters corresponding to two critical stages are set to follow their criticality order. Precisely, in the first iteration, the design hyperparameters of the first and second most critical stages are set to maximize the proportion of ReLUs in the most critical stage. This iterative process executes for $D-1$ iterations (for a network with D stages) and eventually outputs an expression in the form of compound inequalities. The final solution of the compound inequalities outputs a range of values for design hyperparameters to achieve ReLU equalization, and the most critical stage has the highest proportion of ReLUs whereas the least critical stage has the lowest proportion of ReLUs. In this process of redesigning networks, a total number of design hyperparameters is $2D-1$ (D stage compute ratios and $D-1$ $ChMulFact$s) which can be further reduced by setting either depth or width hyperparameters fixed. For example, HRN networks achieve ReLU equalization through width, without changing the depth, and the number of design hyper-parameters is reduced to $D-1$.

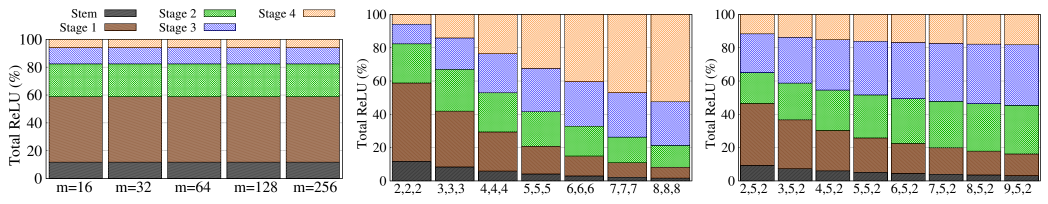
ReLU equalization on four stage networks: We employ DeepReShape on a four-stage network with the given sets of depth and width hyperparameters as shown in Figure 3. We get four pairs of (β, γ) and a range of $a\alpha$ values, as shown in Appendix B.1.Now, for finding the appropriate values of (α, β, and γ), from the above-derived range of (α, β, and γ), we plot the stagewise ReLU distribution for all the four (β, and γ) pairs and vary the α from α=2 to higher values (see Figure 5(c), and Figure 13(a-c) in Appendix D.3). Similar to $StageCh$ networks, the proportion of Stage1 ReLUs starts decreasing until the network's ReLUs get equalized in their order of criticality. Once the network achieves ReLU equalization at a specific value of α, the relative distribution of ReLUs remains stable. That is, unlike the $StageCh$ networks, the proportion of significant ReLUs does not change at higher values of α. This highlight the significance of ReLU equalization from ReLUs' distribution viewpoint. Finally, we take minimum values of α, β, and γ achieving the ReLU equalization, and we get four baseline networks: HRN-7x5x2x, HRN-5x5x3x, HRN-6x6x2x, and HRN-5x7x2x whose α values correspond to the minimum possible value for ReLU equalization (see Appendix D.3 for detailed discussion).

Algorithm 1 DeepReShape

Input: A network Net with D stages S_1, S_2, ..., S_D, C a sorted list of stages from most to least critical stage, stage-compute ratio ϕ_1, ϕ_2, ..., ϕ_D, and stage-wise channel multiplication factor as λ_1, λ_2, ..., $\lambda_{(D-1)}$

Output: A range of possible values of ϕ_1, ϕ_2, ..., ϕ_D and λ_1, λ_2, ..., $\lambda_{(D-1)}$ for ReLU-equalized versions of Net.

```
1: for $i$ = 1 to D-1 do
2:     $S_k$ = C[i]                                  ▷ C[1] is most critical stage
3:     $S_t$ = C[i+1]                                ▷ C[2] is second-most critical stage
4:     #ReLUs($S_k$) $>$ #ReLUs($S_t$)                ▷ Evaluate the compound inequality
5:         $\Longrightarrow \frac{\phi_k \times \left(\prod_{j=1}^{k-1} \lambda_j\right)}{2^{k-1}} > \frac{\phi_t \times \left(\prod_{j=1}^{t-1} \lambda_j\right)}{2^{t-1}}$
6: end for
7: return Possible values of $\phi_1$, $\phi_2$, ...., $\phi_D$ and $\lambda_1$, $\lambda_2$, ...., $\lambda_{(D-1)}$
```



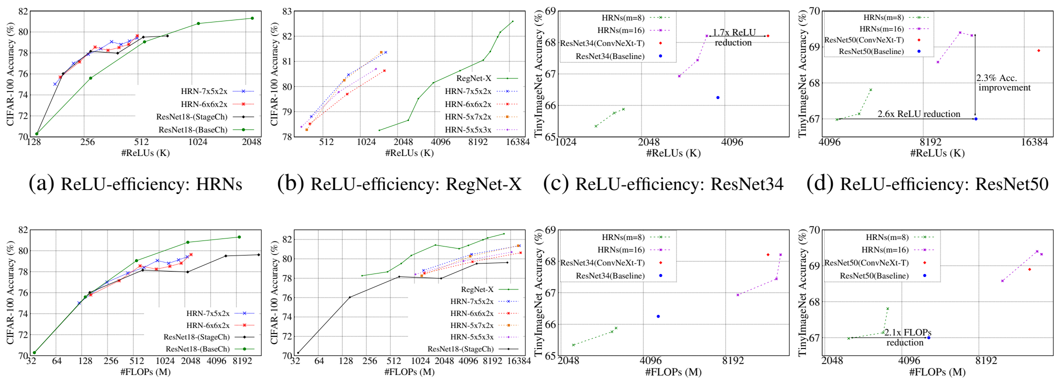
(a) ResNet18(BaseCh) networks (b) ResNet18(StageCh) networks (c) HRN-7x5x2x(Proposed)

Figure 5: Stagewise (relative) distribution of ReLUs for: (a) ResNet18 $BaseCh$ networks; (b) ResNet18 $StageCh$ networks; (c) HRN network with $\alpha \geq 2$ and (β, γ) = (5,2). Values of α in HRN is swept for a holistic characterization. Once HRN network achieves ReLU equalization, increasing α does not change the relative distribution of ReLUs across stages.

4 Experiments

FLOPs-ReLU-Accuracy balance: Since FLOPs are not free (while ReLU is the primary bottleneck) in PI (Garimella et al., 2022), unlike the previous PI-friendly network design methodsGhodsi et al. (2020); Aggarwal et al. (2020); Cho et al. (2022a), we demonstrate both the FLOPs and ReLUs characteristic of HybReNets and compare their efficiency with $BaseCh$ and $StageCh$ networks in Figure 6(a) and Figure 6(d). For HRN networks, we sweep the α=2 to higher values to analyze the complete behavior, especially the model-wise deep double descent (observed in $StageCh$ networks, see Appendix D.2 for a detailed discussion). We make the following observation: (1) HRN networks exceed the ReLU-efficiency of $StageCh$ networks with fewer FLOPs (FLOPs-efficiency approaching that of the $BaseCh$ networks at higher αs); and (2) similar to $StageCh$ networks, HRN networks exhibit model-wise deep double descent; however, with a smaller plateau in the case of HRNs. Thus, it is evident that HRN networks perform better FLOPs-ReLUs-Accuracy tradeoffs compared to the $BaseCh$ and $StageCh$ networks. To understand the mechanism through which HRN networks achieve FLOPs-ReLUs-Accuracy balance, a detailed discussion is included in Appendix D.4. We also compare the ReLU and FLOPs efficiency when ReLU equalization is performed through depth in $BaseCh$ networks, in Appendix C.1. This demonstrates that even when ReLUs' are equalized in $BasCh$ networks, their ReLU-efficiency is inferior to $StageCh$ networks.

Generality of ReLU equalization for designing FLOPs efficient networks: A case study with RegNet and ConvNeXt We now show the generality of ReLU equalization even for designing FLOPs efficient networks, and for this we select state-of-the-art vision models, RegNet (Radosavovic et al., 2020), and ConvNeXt (Liu et al., 2022b). RegNets are the outcome of a semi-automated network design method, parametrized by the stage compute ratio (ϕ_1, ϕ_2, ϕ_3, ϕ_4), base channel count (m), and stagewise channel multiplication factors as $1.5 \leq$(α, β, γ)$\leq$ 3. Likewise, ConvNeXts are the outcome of redesigned ResNet with modified m, and ϕ_1, ϕ_2, ϕ_3, ϕ_4. For instance, for ConvNeXt-T model, the stage compute ratio is changed to [3, 3, 9, 3] from [3, 4, 6, 3], and m=96 from m=64 in ResNet50. Thus, the (relatively)unconstrained design choices for RegNets, and modified depth and width in ConvNeXt models made them appropriate for our case study, and we investigate their impact on ReLU distribution and FLOPs-ReLU-Accuracy balance.



(a) ReLU-efficiency: HRNs (b) ReLU-efficiency: RegNet-X (c) ReLU-efficiency: ResNet34 (d) ReLU-efficiency: ResNet50

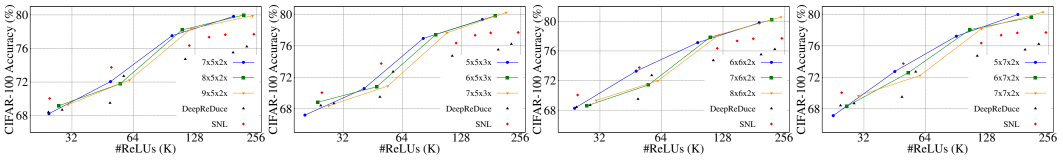
(e) FLOPs-efficiency: HRNs (f) FLOPs-efficiency: RegNet-X (g) FLOPs-efficiency: ResNet34 (h) FLOPs-efficiency: ResNet50

Figure 6: FLOPs-ReLU-Accuracy tradeoff (upper/lower rows the ReLU/FLOPs efficiency): (a)/(e) show the ReLU/ FLOPs efficiency for HRN networks, in contrast with $BaseCh$ and $StageCh$ networks. (b)/(f) compare the ReLU/FLOPs efficiency for RegNet-X, a family of ReLU-equalized networks. *HRNs exceed the ReLU-efficiency of StageCh networks with fewer FLOPs* and achieve FLOPs-ReLU-Accuracy balance. (c,d)/(g,h) compare the ReLU/FLOPs efficiency of HRNs and ConvNeXt-ify version of (ResNet34, ResNet50) with baseline networks, respectively. **HRNs substantially improve the ReLU-efficiency and outperform their baseline and ConvNeXt-ify network**. For ResNet50, HRNs also improve the FLOPs-efficiency significantly.

ReLU distribution for RegNet-X models is shown in Figure 13(f), and *interestingly* ReLU distribution for all the models follow their criticality order. We note that for networks with very low ReLU count criticality order of Stage2 and Stage4 gets interchanged (see Table 5), and following that ReLUs distribution in RegNet-0.2GF and RegNet-0.4GF are different (Stage3 $>$ Stage4 $>$ Stage2 $>$ Stage1) than rest of the RegNet models. Similarly, we compare the ReLUs' distribution of ConvNeXt-ify ResNet34 and ResNet50 with the baseline and HRN networks in Figure 13(d) and Figure 13(e), respectively. Evidently, ConvNeXt equalized the ReLUs' distribution following their criticality order, except the Stage1. Effectively, ConvNeXt increases the proportion of (most)significant ReLUs — from **20.3%** to **30.2%** in ResNet34, and from **21.7%** to **31%** in ResNet50 — and reducing the proportion of less critical ReLUs. Further, we compare the FLOPs-ReLU-Accuracy tradeoffs in RegNets with HRN networks in Figure 6(b,f) and, and ConvNeXt-ify ResNet34 and ResNet50, with the baseline and HRN networks, in Figure 6(c,g) and Figure 6(d,h), respectively. Clearly, while RegNets are FLOPs-efficient, their ReLU-efficiency is substantially inferior to the HRN networks. Likewise, at iso-accuracy on TinyImageNet, HRN achieves 1.7$\times$ ReLU reduction over ConvNeXt-ify ResNet34, and 2.6$\times$ ReLU reduction over ResNet50 baseline. Also, at iso-ReLU count on TinyImageNet, HRN network achieves 2.3% accuracy improvement over the baseline ResNet50. In fact, HRN network outperforms ResNet50 baseline for FLOPs-efficiency and reduces the 2.1$\times$ FLOPs at iso-accuracy. This highlights the significance of ReLU equalization for FLOPs-ReLU-Accuracy balance over a wide range of complexity. We also discuss the potential of ReLU equalization being a unified design principle for both FLOPs and ReLU efficient networks, in contrast with the manual and neural architecture search in Appendix D.5.

DeepReShape with ReLU optimization methods: Given the least fraction of non-critical ReLUs in HRNs, it remains interesting to find additional improvement in their ReLU efficiency when ReLU optimization techniques are applied on top of HRN networks. We apply DeepReDuce (Jha et al., 2021), a coarse-grained ReLU optimization based on stagewise ReLUs' criticality, and compare their performance with state-of-the-art fine-grained ReLU optimization SNL (Cho et al., 2022b) in Figure 7. We also demonstrate the effect of increasing α in HRNs, and made the following observation: (1) HRNs significantly outperform SNL at higher ReLU counts, for example, at an iso-ReLU count of 180K HRN-5x7x2x (with DeepReDuce) outperforms the SNL by **2.35%**; and (2) increasing α does not improve ReLU efficiency, in fact, performs drops at lower ReLU count.

Here, we would like to emphasize that DeepReDuce with ReLU-equalized HRN networks requires only one iteration, in contrast with $D-1$ iterations for DeepReDuce with classical networks having ReLUs' distribution *agnostic* to their critical order, and single iteration often results in sup-optimal



(a) HRN-7x5x2x (b) HRN-5x5x3x (c) HRN-6x6x2x (d) HRN-5x7x2x

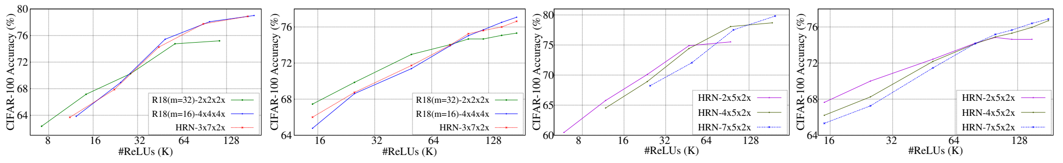
Figure 7: Performance comparison of DeepReDuce Jha et al. (2021) on (proposed)HybReNet with SNL Cho et al. (2022b): **HRNs with DeepReDuce outperform SNL at higher #ReLUs by ∼2.5%**.

ReLU-efficiency. Thus, complexity is substantially reduced, precisely by $(D-1)\times$ for a D network. Now, we investigate the performance drop in HRNs with DeepReDuce at lower ReLU.

Capacity-Criticality Tradeoff: To investigate the performance drop of the aforementioned HRN networks at lower ReLUs, we select three networks with distinct ReLU distribution at the iso-ReLU count (in Table 4) for our case study. In particular, ResNet18(m=32)-2x2x2x has the highest fraction of ReLUs in the least critical Stage (Stage1), ResNet18(m=16)-4x4x4x, stages have equal proportion of ReLUs, and in HRN-3x7x2x distribution of ReLUs follow their criticality order (except the Stage1). We apply DeepReDuce and SNL on these three networks and the results are shown in Figure 8(a) and Figure 8(b), respectively. Evidently, for both DeepReDuce and SNL networks with fewer non-critical ReLUs (Stage1) performs better at higher ReLUs; however, for lower ReLU counts networks with a higher number of critical ReLUs perform better. Further, to lend more strength to the aforementioned observation, we repeat the above experiments with HRN networks. In particular, reducing α values in HRN networks increases the proportion of (non-critical)ReLUs in Stage1 (see Table 5); however, the ReLUs in the remaining stages follow their criticality order. Results are shown in Figure8(c,d) and Figure 12 (in Appendix), and our observations for criticality-capacity tradeoff is consistent in these experiments.

Table 4: A case study for criticality-capacity tradeoff: Networks with distinct ReLUs' distribution, in particular, the proportion of non-critical ReLUs in Stage1, at iso-ReLU.

Model	Acc(C100)	#FLOPs	ReLUs	Stage-wise ReLU breakdown				Stage-wise $\frac{FLOPs}{ReLU}$			
				Stage1-count	Stage2-count	Stage3-count	Stage4-count	Stage1	Stage2	Stage3	Stage4
R18(m=32)-2x2x2x	75.60%	140.9M	278.53K	163840(58.82%)	65536(23.53%)	32768(11.76%)	16384(5.88%)	32	64	128	256
R18(m=16)-4x4x4x	78.16%	661.2M	278.53K	81920(29.41%)	65536(23.53%)	65536(23.53%)	65536(23.53%)	16	64	256	1024
HRN-3x7x2x	78.02%	465.7M	260.10K	81920(31.50%)	49152(18.90%)	86016(33.07%)	43008(16.54%)	16	48	336	672

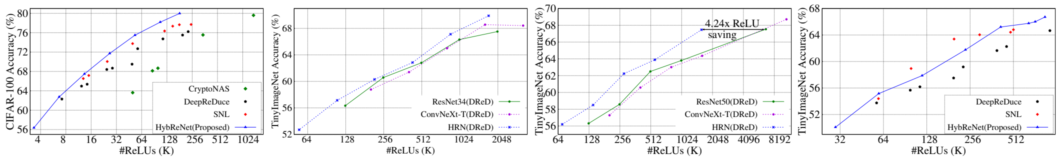


(a) DeepReDuce at iso-ReLU (b) SNL at iso-ReLU (c) DeepReDuce for HRN-5x7x2x (d) SNL for HRN-5x7x2x

Figure 8: Capacity-Criticality Tradeoff for both coarse/fine-grained ReLU optimization (DeepReDuce/SNL): (a)/(b) show the tradeoff for DeepReDuce/SNL on iso-ReLU networks with distinct ReLU distribution; (c)/(d) show the tradeoff for HRN networks with distinct fraction of ReLUs in Stage1, and ReLUs' distribution in the remaining stages follows their criticality order. **HRN-3x7x2x exhibits similar ReLU efficiency with 30% fewer FLOPs**, compared to ResNet18(m=16)-4x4x4x.

Note that, to achieve a very low ReLU count, a network with higher number of non-critical ReLUs needs to drop fewer critical ReLUs, in contrast with a network with lesser non-critical ReLUs that needs to drop more critical ReLUs. Moreover, as explained in Yosinski et al. (2014), neurons in the middle layers (i.e., Stage2 and Stage3) exhibit fragile co-adaption, which is harder to re-learn; thus, dropping more ReLUs from these stages, in networks with fewer non-critical ReLUs, would disrupt the fragile co-adaption and incur a significant performance penalty.

ReLU-reuse: We propose ReLU-reuse, a (structured)channel-wise ReLU dropping, for a better alternative of conventional scaling used in DeepReDuce to achieve a very low ReLU count. Inspired by Gao et al. (2019), we first implement a naive way of dropping channel-wise ReLUs by a factor of N where feature maps are divided into N groups and ReLUs are employed only in the last group, as shown in Figure 15. However, we note a significant accuracy drop, even when we use 1×1 convolution for cross-channel interaction in feature subspace, at higher N caused by the higher



(a) HRNs on CIFAR-100 (b) ResNet34 comparison (c) ResNet50 comparison (d) HRNs on TinyImageNet

Figure 9: HRN with DeepReDuce on CIFAR-100 (Figure (a)), and TinyImageNet (Figure (d)). We compare HRNs with DeepReDuce on ResNet34 and ResNet50 in Figure (c) and (d), respectively.

number of divisions in feature-maps; thus, losing cross-channel information (see ablation study in Table 9 in Appendix E). To remedy this, we devise a mechanism where the number of divisions in feature maps is independent of the ReLU reduction factor N. In particular, similar to Gao et al. (2019), one forth of channels is used for feature reuse, Nth fraction of feature-maps are processed with convolution followed by ReLUs, and the remaining fraction of feature-maps are processed only with convolution. Note that using the ReLUs in the last group of feature maps increases the effective receptive since those neurons can see almost the entire input feature maps. We demonstrate the scalability of ReLU-reuse, in contrast with reshaping in DeepReDuce, on ResNet $BaseCh$ and HRN models in Figure 16 and 17. Further, we incorporate the ReLU-reuse in DeepReDuce pipeline (Algorithm 2), and results are shown in Figure 18. We note an accuracy gain upto **3%**, at iso-ReLU.

Putting it all together: We apply DeepReDuce with ReLU-reuse on HRN networks and compare their performance in Figure 9. First, we compare the ReLU-accuracy tradeoff on CIFAR-100 and the results are shown in Figure 9(a). At higher ReLU count, HRN with DeepReDuce significantly outperforms the SNL; in particular, HRN-5x7x2x at 180K ReLUs achieve a 2.35% accuracy gain. We also compare the performance with DeepReDuce on baseline and ConvNeXt-ify ResNet34/ResNet50 (on TinyImageNet) in Figure 9(b)/(c), and note that HRNs perform better on higher as well as lower ReLU counts. HRN saves **4.2**$\times$ ReLU at iso-ReLU for ResNet50 on TinyImagNet. Further, we compare the HRNs with SNL on TinyImageNet in Figure 9(d). We note that at higher ReLUs, and very low ReLU counts, HRNs outperform SNL; however, at intermediate ReLUs counts SNL is superior. Here, we would like to emphasize that SNL drops the (unstructured)pixel-wise ReLUs, in contrast with DeepReDuce with ReLU-reuse (structured ReLU dropping). Thus, on a complex dataset, SNL can find the better spots for dropping ReLUs at intermediate ReLU counts.

5 Related Work

PI-amenable network optimization: Delphi (Mishra et al., 2020) and SAFENet (Lou et al., 2021) substitute the ReLUs with low-degree polynomials, while DeepReDuce (Jha et al., 2021) is a coarse-grained ReLU optimization and drops ReLUs layerwise. SNL (Cho et al., 2022b) is a state-of-the-art fine-grained ReLU optimization, and drops the pixelwise ReLUs. CryptoNAS (Ghodsi et al., 2020) and Sphynx Cho et al. (2022a) use neural architecture search and employ constant number of ReLUs per layer for designing ReLU-efficient networks. In contrast, we demonstrated that distributing the ReLUs in their criticality order is better for ReLU-efficiency and FLOPs-ReLU-accuracy balance.
Benefits of width: Li et al. (2018) studied the effect of width on the smoothness of loss surface and showed that increasing the width helps prevent chaotic behavior in loss landscape. Golubeva et al. (2021) decoupled the effect of increasing width and over-parameterization, and showed that the network's width is the primary factor for the network's predictive performance. Liu et al. (2022a) showed that when fixing the depth, increasing the widening factor improves the adversarial robustness. Mirzadeh et al. (2022) demonstrated the benefit of width for mitigating catastrophic forgetting.

6 Conclusion

In this work, we study the benefits of width for ReLU-efficiency and proposed ReLU equalization as a network design principle to allocate ReLUs appropriately in the network. We use a set of design principles, DeepReShape, to design a family of ReLU-efficient networks HybReNet. We demonstrated their benefits in terms of FLOPs-ReLU-Accuracy balance in conjunction with the ReLU optimization methods.

REFERENCES

Anshul Aggarwal, Trevor E Carlson, Reza Shokri, and Shruti Tople. Soteria: In search of efficient neural networks for private inference. *arXiv preprint arXiv:2007.12934*, 2020.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 2019.

Minsu Cho, Zahra Ghodsi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Sphynx: Relu-efficient network design for private inference. *IEEE Security & Privacy*, 2022a.

Minsu Cho, Ameya Joshi, Siddharth Garg, Brandon Reagen, and Chinmay Hegde. Selective network linearization for efficient private inference. In *International Conference on Machine Learning*, 2022b.

Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip HS Torr. Res2net: A new multi-scale backbone architecture. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

Karthik Garimella, Zahra Ghodsi, Nandan Kumar Jha, Siddharth Garg, and Brandon Reagen. Characterizing and optimizing end-to-end systems for private inference. *arXiv preprint arXiv:2207.07177*, 2022.

Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. CryptoNAS: Private inference on a relu budget. In *Advances in Neural Information Processing Systems*, 2020.

Zahra Ghodsi, Nandan Kumar Jha, Brandon Reagen, and Siddharth Garg. Circa: Stochastic relus for private deep learning. In *Advances in Neural Information Processing Systems*, 2021.

Anna Golubeva, Guy Gur-Ari, and Behnam Neyshabur. Are wider nets better given the same number of parameters? In *International Conference on Learning Representations*, 2021.

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*, 2016.

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

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*, 2019.

Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. DeepReDuce: Relu reduction for fast private inference. In *International Conference on Machine Learning*, 2021.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, 2018.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7, 2015.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 2018.

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

Hao Liu, Minshuo Chen, Siawpeng Er, Wenjing Liao, Tong Zhang, and Tuo Zhao. Benefits of overparameterized convolutional residual networks: Function approximation under smoothness constraint. In *International Conference on Machine Learning*, 2022a.

Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Qian Lou, Yilin Shen, Hongxia Jin, and Lei Jiang. SAFENet: Asecure, accurate and fast neu-ral network inference. *International Conference on Learning Representations*, 2021.

Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Huiyi Hu, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Wide neural networks forget less catastrophically. In *International Conference on Machine Learning*, 2022.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium*, 2020.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021.

Daniel Park, Jascha Sohl-Dickstein, Quoc Le, and Samuel Smith. The effect of network width on stochastic gradient descent and generalization: an empirical study. In *International Conference on Machine Learning*, 2019.

Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2020.

Gowthami Somepalli, Liam Fowl, Arpit Bansal, Ping Yeh-Chiang, Yehuda Dar, Richard Baraniuk, Micah Goldblum, and Tom Goldstein. Can neural nets learn the same model twice? investigating reproducibility and double descent from the decision boundary perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019.

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

Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. In *IEEE Symposium on Security and Privacy*, 2021.

Yongqin Wang, G Edward Suh, Wenjie Xiong, Benjamin Lefaudeux, Brian Knott, Murali Annavaram, and Hsien-Hsin S Lee. Characterization of mpc-based private inference for transformer-based models. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

Antoine Yang, Pedro M. Esperana, and Fabio M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020.

Leon Yao and John Miller. Tiny imagenet classification with convolutional neural networks. *CS 231N*, 2015.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint*, 2016.

A STAGEWISE RELUS CRITICALITY WITH WIDTH ARGUMENTATION

Table 5: Stagewise ReLU criticality of HRN networks (derived from ResNet18) with α values ranging from 2 to 7, on CIFAR-100 dataset. The HRNs with minimum α, β, and γ requiring for ReLU-equalization (in all the stages) are in **bold**. The criticality metric values (C_k) for all the stages are computed according to the method described in Jha et al. (2021). Higher criticality metric represent stage with more critical ReLUs. We note that the *criticality orders of all the HRN networks, except the smallest network with α=2, are same as that in original ResNet18* (i.e., $S_3 > S_2 > S_4 > S_1$).

Networks	Stage1				Stage2				Stage3				Stage4			
	#ReLUs	Acc(%)	KD(%)	C_k	#ReLUs	Acc(%)	KD(%)	C_k	#ReLUs	Acc(%)	KD(%)	C_k	#ReLUs	Acc(%)	KD(%)	C_k
HRN-2x7x2x	81.92K	52.14	53.39	0.00	32.77K	61.63	61.59	6.42	57.34K	68.44	69.82	12.37	28.67K	62.15	63.40	7.91
HRN-3x7x2x	81.92K	51.61	53.29	0.00	49.15K	64.46	65.26	9.11	86.02K	69.88	70.77	12.80	43.01K	63.10	64.17	8.36
HRN-4x7x2x	81.92K	51.28	49.42	0.00	65.54k	65.93	66.47	12.72	114.69K	70.94	72.16	16.32	57.34K	63.70	64.77	11.56
HRN-5x7x2x	**81.92K**	**49.82**	**48.36**	**0.00**	**81.92K**	**66.17**	**67.59**	**14.13**	**143.36K**	**71.40**	**72.18**	**16.83**	**71.68K**	**64.10**	**65.35**	**12.60**
HRN-6x7x2x	81.92K	51.23	48.48	0.00	98.30K	66.88	68.06	14.20	172.03K	71.86	72.73	16.91	86.02K	64.15	65.75	12.64
HRN-7x7x2x	81.92K	50.11	52.40	0.00	114.69K	66.92	68.29	11.40	200.70K	71.69	73.16	14.32	100.35K	63.82	65.53	9.51
HRN-2x6x2x	81.92K	52.29	53.19	0.00	32.77K	61.62	62.00	6.90	49.15K	67.36	69.51	12.43	24.58K	61.64	63.25	8.04
HRN-3x6x2x	81.92K	52.50	52.80	0.00	49.15K	64.50	65.64	9.78	73.73K	68.61	70.96	13.44	36.86K	62.77	64.09	8.77
HRN-4x6x2x	81.92K	53.23	53.32	0.00	65.54K	65.74	66.03	9.48	98.30K	70.47	71.54	13.22	49.15K	63.59	64.82	8.76
HRN-5x6x2x	81.92K	50.79	51.64	0.00	81.92K	66.89	67.27	11.48	122.88K	70.33	71.50	14.18	61.44K	63.97	64.94	9.97
HRN-6x6x2x	**81.92K**	**50.01**	**50.59**	**0.00**	**98.30K**	**66.57**	**67.94**	**12.58**	**147.46K**	**71.18**	**72.59**	**15.51**	**73.73K**	**64.13**	**65.39**	**10.95**
HRN-7x6x2x	81.92K	51.01	49.64	0.00	114.69K	66.74	68.57	13.58	172.03K	71.84	72.84	16.18	86.02K	64.54	65.16	11.36
HRN-2x5x2x	81.92K	52.03	53.05	0.00	32.77K	61.60	61.76	6.82	40.96K	66.64	68.29	11.75	20.48K	61.02	62.58	7.71
HRN-3x5x2x	81.92K	53.43	52.61	0.00	49.15K	64.57	65.71	9.97	61.44K	68.40	69.93	12.98	30.72K	62.32	63.42	8.51
HRN-4x5x2x	81.92K	52.65	52.33	0.00	65.54K	65.60	66.89	10.86	81.92K	69.81	70.85	13.61	40.96K	63.14	63.94	8.95
HRN-5x5x2x	81.92K	49.15	51.16	0.00	81.92K	66.26	67.47	11.98	102.40K	70.15	71.69	14.85	51.20K	63.55	64.67	10.26
HRN-6x5x2x	81.92K	49.06	52.10	0.00	98.30K	66.56	68.08	11.59	122.88K	71.33	71.85	14.10	61.44K	63.59	64.89	9.59
HRN-7x5x2x	**81.92K**	**51.58**	**51.93**	**0.00**	**114.69K**	**66.94**	**67.89**	**11.45**	**143.36K**	**70.79**	**72.87**	**14.79**	**71.68K**	**64.02**	**65.23**	**9.86**
HRN-2x5x3x	81.92K	52.36	53.68	0.00	32.77K	61.39	61.30	5.97	40.96K	66.78	68.17	11.17	30.72K	62.01	63.83	7.99
HRN-3x5x3x	81.92K	51.05	52.89	0.00	49.15K	64.64	65.10	9.30	61.44K	68.87	70.14	12.93	46.08K	63.66	64.32	8.74
HRN-4x5x3x	81.92K	51.57	50.62	0.00	65.54K	65.66	66.06	11.52	81.92K	69.12	70.13	14.33	61.44K	63.64	65.58	11.21
HRN-5x5x3x	**81.92K**	**50.22**	**52.41**	**0.00**	**81.92K**	**66.42**	**67.55**	**11.12**	**102.40K**	**70.15**	**70.97**	**13.42**	**76.80K**	**64.21**	**65.59**	**9.73**
HRN-6x5x3x	81.92K	50.28	50.45	0.00	98.30K	65.95	67.61	12.45	122.88K	70.68	71.29	14.88	92.16K	64.37	65.87	11.23
HRN-7x5x3x	81.92K	50.12	50.31	0.00	114.69K	66.85	67.95	12.66	143.36K	71.20	71.87	15.23	107.52K	64.72	65.58	11.01
HRN-2x9x2x	81.92K	51.86	53.22	0.00	32.77K	61.13	61.65	6.60	73.73K	69.46	70.28	12.63	36.86K	62.53	64.25	8.57
HRN-2x6x3x	81.92K	52.75	52.85	0.00	32.77K	61.33	61.44	6.73	49.15K	67.36	68.76	12.11	36.86K	62.69	64.59	9.12

We compute the stagewise criticality metric for all the HRN networks derived from ResNet18 in Table 5. The same is computed for the ResNet18 $BaseCh$ and $StageCh$ networks in Table 6. *Interestingly*, the criticality order of vanilla ResNet18 remains preserved in all the HRNs, except for the HRNs with α=2 (HRN-2x5x3x, HRN-2x5x2x, HRN-2x6x2x, and HRN-2x7x2x). Precisely, the criticality order of Stage2 and Stage3 get shuffled, and it changed to Stage3 $>$ Stage4 $>$ Stage2 $>$ Stage1 (from Stage3 $>$ Stage2 $>$ Stage4 $>$ Stage1), while the order of most and least critical stages remains same. For this change order of criticality, we again compute the values of α, β, and γ using the DeepReShape algorithm 1 while keeping the fraction of ReLUs in Stage1 highest. In this way, we get two HRN-2x6x3x and HRN-2x9x3x where β and γ correspond to the minimum value satisfying ReLU equalization in all the stages, except Stage1. We also compute the stagewise criticality for these HRNs in the bottom rows of Table 5.

Table 6: Stagewise ReLU criticality of $BaseCh$ and $StageCh$ networks derived from ResNet18 (R18), on CIFAR-100 dataset. The criticality metric values (C_k) for all the stages are computed according to the method described in Jha et al. (2021). We note that the *criticality order of all the $BaseCh$ and $StageCh$ networks are same as that in original ResNet18 (i.e., $S_3 > S_2 > S_4 > S_1$).*

Networks	Stage1				Stage2				Stage3				Stage4			
	#ReLUs	Acc(%)	KD(%)	C_k	#ReLUs	Acc(%)	KD(%)	C_k	#ReLUs	Acc(%)	KD(%)	C_k	#ReLUs	Acc(%)	KD(%)	C_k
R18(m=16)-2x2x2x	81.92K	52.08	52.67	0.00	32.77K	61.24	62.10	7.39	16.38K	63.00	64.64	9.84	8.19K	58.09	59.70	6.07
R18(m=32)-2x2x2x	163.84K	59.19	60.19	0.00	65.54K	65.91	66.47	4.69	32.77K	65.7	67.28	5.55	16.38K	60.48	62.22	1.67
R18(m=64)-2x2x2x	327.68K	62.65	63.13	0.00	131.07K	67.18	68.32	3.69	65.54K	68.75	70.29	5.34	32.77K	62.63	63.47	0.27
R18(m=128)-2x2x2x	655.36K	62.34	64.15	0.00	262.14K	69.28	70.56	4.34	131.07K	71.25	72.04	5.61	65.54K	63.59	64.58	0.32
R18(m=256)-2x2x2x	1310.72K	64.81	65.22	0.00	524.29K	71.95	72.43	4.65	262.14K	72.69	73.77	5.79	131.07K	64.79	65.77	0.39
R18(m=16)-2x2x2x	81.92K	52.08	52.67	0.00	32.77K	61.24	62.10	7.39	16.38K	63.00	64.64	9.84	8.19K	58.09	59.70	6.07
R18(m=16)-3x3x3x	81.92K	52.77	53.07	0.00	49.15K	64.93	65.67	9.59	36.86K	66.23	67.96	11.57	27.65K	61.74	63.43	8.21
R18(m=16)-4x4x4x	81.92K	52.19	52.20	0.00	65.54K	65.62	66.22	10.46	65.54K	67.82	69.16	12.66	65.54K	63.52	65.46	9.89
R18(m=16)-5x5x5x	81.92K	50.38	50.65	0.00	81.92K	66.10	66.63	11.74	102.40K	70.17	70.64	14.46	128.00K	64.86	65.43	10.52
R18(m=16)-6x6x6x	81.92K	50.60	51.53	0.00	98.30K	66.74	67.11	11.30	147.46K	70.67	72.09	14.49	221.18K	65.22	66.43	10.21
R18(m=16)-7x7x7x	81.92K	50.93	49.07	0.00	114.69K	66.59	67.89	13.50	200.70K	72.08	73.33	16.74	351.23K	65.95	67.88	12.48

B RELU EQUALIZATION AND BUILDING BLOCKS IN HYBRENET

B.1 RELU EQUALIZATION ON A FOUR STAGE NETWORK

In order to achieve ReLU equalization (shown in Figure10), we employ DeepReShape on a (standard) four-stage network shown in Figure 3. In particular, given a D stages input network with ϕ_1, ϕ_2, ..., ϕ_D as stage compute ratios, and λ_1. λ_2, ..., $\lambda_{(D-1)}$ as the stagewise channel multiplication factors, first we find the stagewise criticality order similar to Jha et al. (2021). Given the criticality

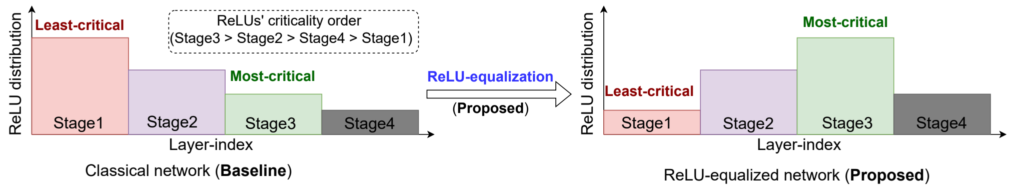


Figure 10: ReLUs' distribution in classical networks (e.g., ResNet) is *agnostic* to their criticality order, and through ReLU equalization network's ReLUs are redistributed in their order of criticality.

order (from most critical to least critical) $S_3 > S_2 > S_4 > S_1$ (see Table 6). DeepReShape 1 outputs following compound inequalities.

$$\#ReLUs(S_3) > \#ReLUs(S_2) > \#ReLUs(S_4) > \#ReLUs(S_1)$$

$$\implies \phi_3\left(\frac{\alpha\beta}{16}\right) > \phi_2\left(\frac{\alpha}{4}\right) > \phi_4\left(\frac{\alpha\beta\gamma}{64}\right) > \phi_1$$

$$\text{ReLU equalization through depth } (\alpha=\beta=\gamma=2): \implies \frac{\phi_3}{4} > \frac{\phi_2}{2} > \frac{\phi_4}{8} > \phi_1$$

$$\text{ReLU equalization through width } (\phi_1=\phi_2=\phi_3=\phi_4=2, \text{ and } \alpha\geq 2, \beta\geq 2, \gamma\geq 2):$$

$$\implies \frac{\alpha\beta}{16} > \frac{\alpha}{4} > \frac{\alpha\beta\gamma}{64} > 1 \implies \alpha\beta > 16, \alpha > 4, \alpha\beta\gamma > 64, \beta > 4, \beta\gamma < 16, \text{ and } \gamma < 4$$

Solving the above compound inequalities provides the following (β,γ) pairs and the range of α :

The (β,γ) pairs are: $(5,2)$ & $\alpha\geq 7$; $(5,3)$ & $\alpha\geq 5$; $(6,2)$ & $\alpha\geq 6$; $(7,2)$ & $\alpha\geq 5$

Finally, we get four pairs of (β, γ) and a range of α which satisfies the conditions for ReLU equalization. We select the minimum values of α, β, and γ for two reasons: (1) once the network achieves ReLU equalization, the (relative)distribution of ReLUs becomes stable and does not change with increasing α; and (2) increasing alpha in HRNs does not improve the ReLU-accuracy tradeoff when ReLU optimization is applied, rather it causes the poor ReLU-accuracy tradeoff at lower ReLU counts (see Figure 7).

B.2 Architectural Building Blocks in HybReNet

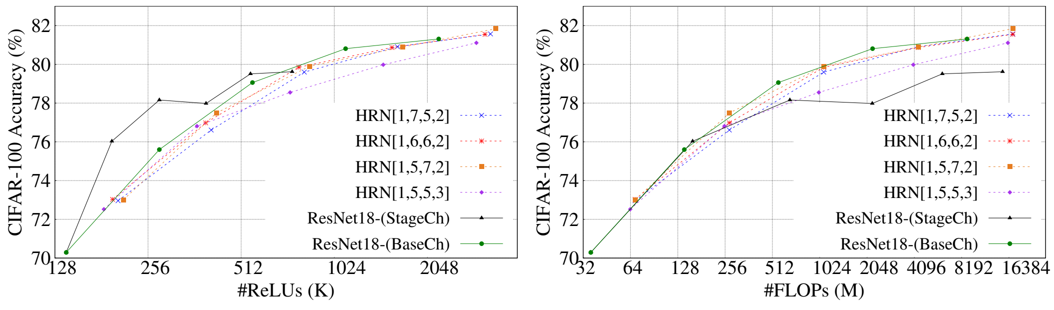
Table 7: Comparison of the building blocks of the (*proposed*) HybReNet with ResNet/WideResNet. Number of channels in ResNet models is progressively multiplied by 2× in each stage (except Stage1); whereas, in WideResNet models the number of channels is further multiplied by a factor of k in all the layers (e.g., k=10 in WRN-22x10). In contrast, channels in HybReNet is *heterogeneously* increased by a factor of α, β, and γ in Stage2, Stage3, and Stage4 (respectively).

Stages	output size	ResNet	WideResNet	HybReNet(**Proposed**)
Stem	$d_{in}\times d_{in}$	[3×3, m]	[3×3, m]	[3×3, m]
Stage1	$d_{in}\times d_{in}$	$\begin{bmatrix} 3\times3, m \\ 3\times3, m \end{bmatrix}\times\phi_1$	$\begin{bmatrix} 3\times3, m\times k \\ 3\times3, m\times k \end{bmatrix}\times\phi_1$	$\begin{bmatrix} 3\times3, m \\ 3\times3, m \end{bmatrix}\times\phi_1$
Stage2	$\frac{d_{in}}{2}\times\frac{d_{in}}{2}$	$\begin{bmatrix} 3\times3, 2m \\ 3\times3, 2m \end{bmatrix}\times\phi_2$	$\begin{bmatrix} 3\times3, 2m\times k \\ 3\times3, 2m\times k \end{bmatrix}\times\phi_2$	$\begin{bmatrix} 3\times3, \alpha m \\ 3\times3, \alpha m \end{bmatrix}\times\phi_2$
Stage3	$\frac{d_{in}}{4}\times\frac{d_{in}}{4}$	$\begin{bmatrix} 3\times3, 4m \\ 3\times3, 4m \end{bmatrix}\times\phi_3$	$\begin{bmatrix} 3\times3, 4m\times k \\ 3\times3, 4m\times k \end{bmatrix}\times\phi_3$	$\begin{bmatrix} 3\times3, \beta(\alpha m) \\ 3\times3, \beta(\alpha m) \end{bmatrix}\times\phi_3$
Stage4	$\frac{d_{in}}{8}\times\frac{d_{in}}{8}$	$\begin{bmatrix} 3\times3, 8m \\ 3\times3, 8m \end{bmatrix}\times\phi_4$		$\begin{bmatrix} 3\times3, \gamma(\alpha\beta m) \\ 3\times3, \gamma(\alpha\beta m) \end{bmatrix}\times\phi_4$
FC	1×1	$[\frac{d_{in}}{8}\times\frac{d_{in}}{8}, 8m]$	$[\frac{d_{in}}{8}\times\frac{d_{in}}{8}, 4m\times k]$	$[\frac{d_{in}}{8}\times\frac{d_{in}}{8}, \gamma(\alpha\beta m)]$

C Additional Experimental Results

C.1 ReLU Equalization Through Depth in $BaseCh$ Networks

ReLU equalization through the width in HRNs has two (simultaneous)effects: first, it increases the network's complexity per unit of nonlinearity (i.e., parameters and FLOPs per ReLU); and second, the distribution of ReLUs follows their criticality order. Now, to decouple these two effects and study their significance individually, we perform ReLU equalization through depth and increase the base channel counts to increase the parameters and FLOPs per ReLU.



(a) ReLU-efficiency (b) FLOPs-efficiency

Figure 11: ReLU and FLOPs efficiency when network's ReLU (in ResNet18) is equalized by setting the stage compute ratios and while complexity per units of nonlinearity is increased by augmenting $m \in \{16, 32, 64, 128, 256\}$.

Similar to the baseline network in HRNs, we use the classical ResNet18 with m=16 (and α=β=γ=2); however, now, stage compute ratios (ϕ_1, ϕ_2, ϕ_3, and ϕ_4) are the design hyperparameters. We apply the Algorithm 1 for ReLU equalization, and after solving solving the compound inequalities, we get the depth hyperparameters corresponding to the minimum values enabling ReLU equalization as (ϕ_1, ϕ_2, ϕ_3, ϕ_4) $\in$ {(1,5,5,3); (1,5,7,2); (1,6,6,2); and (1,7,5,2)}. Note that the network's global depth (sum of all the stage compute ratio) for all the networks are 14. Now, we increase the parameters and FLOPs per unit of ReLU by sweeping m={16, 32, 64, 128, and 256}. The experimental results are shown in Figure 11 where we compare the ReLU and FLOPs efficiency with $BaseCh$ and $StageCh$ networks. As evident from the plots, both the ReLU-efficiency and FLOPs-efficiency of the above-derived networks are either similar or even worst. For example, HRN[1,5,5,3] exhibits inferior ReLU/FLOPs-efficiency at higher ReLU/FLOPs count, compared to the $BaseCh$ networks. Thus, decoupling the effect of ReLU equalization and higher complexity per unit of nonlinearity helps us understand the significance of ReLU-equalization through width, in particular by changing the width hyperparameters α, β, and γ.

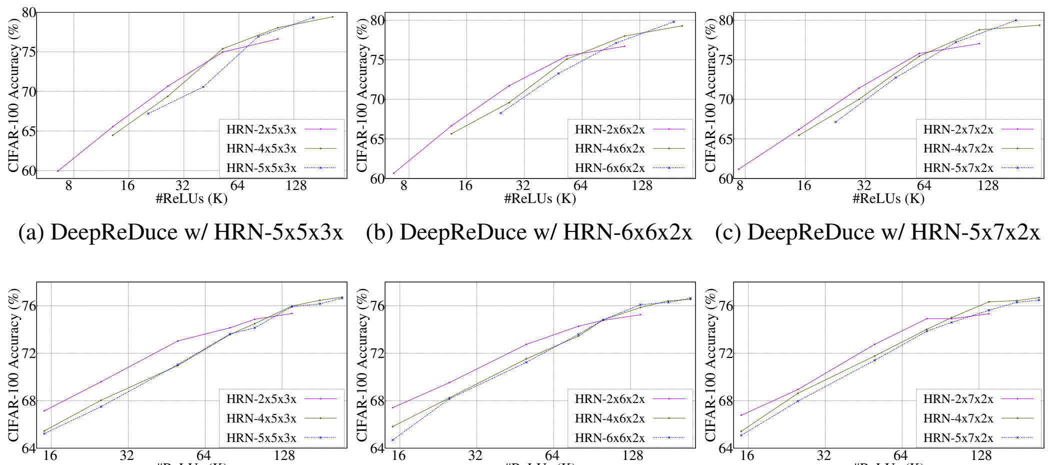
C.2 Additional Results for Criticality-Capacity Tradeoff

In addition to the results shown in Figure 8 for the "Criticality Capacity Tradeoff", we repeat the same experiments on the other HRNs and the results shown in Figure 12. For each set of experiments on DeepReDuce and SNL, we select three HRN networks with reduced values of α, which in turn increases the fraction of ReLUs in Stage1 (see Table 5). For instance, HRN-6x6x2x, HRN-4x6x2x, and HRN-2x6x2x have the Stage1 fraction of ReLUs as 20.4%, 27.8%, and 43.5% (respectively). The observations made in Figure 8 for both DeepReDuce and SNL are consistent with that in Figure 12. For example, at higher ReLUs HRN-6x6x2x and HRN-4x6x2x outperform the HRN-2x6x2x, while at lower ReLU counts, the latter consistency outperforms the former.

D Discussion

D.1 Stagewise Channel Multiplication Factors in Classical Networks

Conventionally, when the spatial size of feature maps is downsampled by a factor of two, then to circumvent the *representational bottleneck* filter count is also doubled (Szegedy et al., 2016).



(a) DeepReDuce w/ HRN-5x5x3x (b) DeepReDuce w/ HRN-6x6x2x (c) DeepReDuce w/ HRN-5x7x2x

(d) SNL w/ HRN-5x5x3x (e) SNL w/ HRN-6x6x2x (f) SNL w/ HRN-5x7x2x

Figure 12: Capacity-Criticality Tradeoff in HRN networks for coarse/fine-grained ReLU optimization DeepReDuce/SNL. HRN networks with decreasing value of α has higher proportion of Stage1 (non-critical) ReLUs.

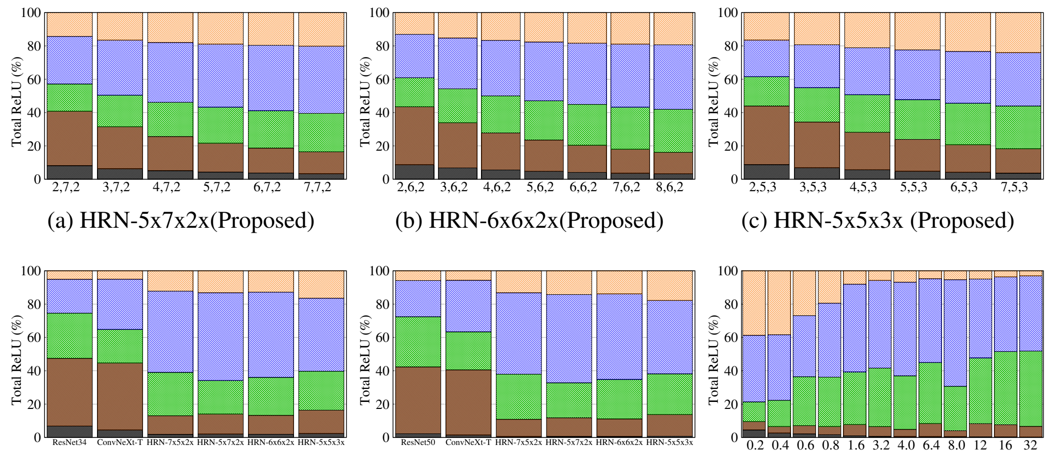
In other words, in almost all the classical networks, the stagewise channel multiplication factors (shown in Figure 3) are set as $\alpha = \beta = \gamma = 2$. Nonetheless, we note that authors in Radosavovic et al. (2020) varied the stagewise channel multiplication factors from 1.5 to 3 for designing FLOPs-efficient networks, and found value ≈ 2.5 works best for FLOPs-efficient networks. The primary reason for being conservative about the stagewise multiplication factor is the FLOPs-efficiency, which the conventional network design paradigms strive for. Precisely, as shown in Table 2, FLOPs has a quadratic dependency on the channel count in conjunction with the multiplicative effect of stagewise channel multiplication factors. Consequently, a slight increase in stagewise multiplication factors can cause a significant increase in overall FLOPs count.

D.2 Accuracy Saturation in $StageCh$ Networks

In contrast with Figure 4(b), we observe a *distinct* trend in ReLU-accuracy trade-off when models' width increases. Precisely, with increasing α, β, and γ, accuracy first increases, and then it gets saturated, and again at higher ReLU count it increases. This trend is more evident in models with smaller depth (e.g., ResNet18); however, it vanishes in deeper models such as ResNet56 where performance does not improve after saturation. We note that the trend of ReLU-accuracy saturation in 4(c) can be explained by the "model-wise deep double descent" phenomenon which is more pronounced with higher label noise (Belkin et al., 2019; Nakkiran et al., 2021; Somepalli et al., 2022). To be specific, in the presence of label noise, a U-shaped curve appears in the classical (under-parameterized) regime due to a bias-variance tradeoff, and in the over-parameterized regime accuracy again improves due to the regularization enabled by the strong inductive bias of network. In contrast, with zero label noise the test error plateaus around the interpolation threshold — similar to the trend shown in Figure 4(c) – and hence, instead of a U-shaped curve it remains flat.

D.3 Additional Results for Stagewise ReLU Distribution in HRN networks

Why increasing α beyond ReLU equalization does not change the relative distribution of ReLU? Recall that scaling the channels with a constant factor does not change the distribution of ReLUs (Figure 5(a)) since it alters the ReLU count of all the stages by same degree (Table 2). Likewise, we notice that increasing α augments the ReLU count of all the stages, except Stage1, by the same degree which makes the (relative) distribution of ReLUs stable at higher αs.



(a) HRN-5x7x2x(Proposed) (b) HRN-6x6x2x(Proposed) (c) HRN-5x5x3x (Proposed)

(d) ResNet34: ConvNeXt & HRNs (e) ResNet50: ConvNeXt & HRNs (f) RegNet-X

Figure 13: ((a)-(c)): Stagewise ReLU distribution in HRN networks where α's are swept from α=2 to higher values in order to perform a holistic characterization. Stagewise ReLU distribution for: (d) ConvNeXt-ify Liu et al. (2022b) version and HRNs with ResNet34; (e) ConvNeXt-ify version and HRNs with ResNet50; and (f) RegNet-X Radosavovic et al. (2020) networks, a semi-automated network design paradigm that outputs ReLU-equalized network.

D.4 How HybReNets Achieves FLOP-ReLU-Accuracy Balance?

As shown in derivation steps for ReLU equalization on a four-stage network in Appendix B.1, HRN networks have (β, γ) pairs with fixed values since it bounds these values as $\beta\gamma < 16$ and $\gamma < 4$. These bounds, especially $\gamma < 4$, prohibit the growth of FLOPs in deeper stages. In contrast, $StageCh$ networks have homogeneous sets of α, β, and γ; thus FLOPs grow rapidly in the deeper layer. As shown in Table 3, normalized FLOPs in Stage3 and Stage4 of ResNet18 is $\frac{\alpha^2\beta^2}{16}$ and $\frac{\alpha^2\beta^2\gamma^2}{64}$ (respectively). Thus, at γ=2, the #FLOPs in Stage3 and Stage4 are equal which is evident from the (normalized)stagewise FLOPs ratio for HRN-5x7x2x, HRN-7x5x2x, and HRN-6x6x2x networks, as shown in Table 8. Notice that γ values are restricted as $\gamma < 4$ to make the ReLU count of Stage4 lower than that of Stage3 (the most critical stage) which in turn restricts the FLOPs count of Stage4. Nonetheless, restricting α and β values can further reduce the network's FLOPs but it also reduces the ReLU count in Stage3 ($\frac{\alpha\beta}{16}$); thus, the proportion of most significant ReLUs in the network would become lower. In conclusion, the criticality-aware network design prohibits the superfluous FLOPs (of $StageCh$ networks) and maximizes the utilization of the network's FLOPs for a given ReLU count.

Table 8: Normalized (stagewise) FLOPs and ReLU count for ResNet18 $StageCh$ networks (upper-table) and HRN networks (lower-table). The least and most critical ReLUs are colored in red and blue, respectively. Evidently, in constrast with $StageCh$ network, **ReLU-equalization in HRNs restrict the growth of FLOPs in deeper layers** and networks achieve ReLU efficiency at par with $StageCh$ network; however, with fewer FLOPs. This way HRN networks achieve FLOPs-ReLU-accuracy balance.

	(α,β,γ)=2				(α,β,γ)=3				(α,β,γ)=4				(α,β,γ)=6			
	Stage1	Stage2	Stage3	Stage4	Stage1	Stage2	Stage3	Stage4	Stage1	Stage2	Stage3	Stage4	Stage1	Stage2	Stage3	Stage4
FLOPs	64	64	64	64	64	144	324	729	64	256	1024	4096	64	576	5184	46656
ReLUs	64	32	16	8	64	48	36	27	64	64	64	64	64	96	144	216
	HRN-5x7x2x				HRN-7x5x2x				HRN-6x6x2x				HRN-5x5x3x			
	Stage1	Stage2	Stage3	Stage4	Stage1	Stage2	Stage3	Stage4	Stage1	Stage2	Stage3	Stage4	Stage1	Stage2	Stage3	Stage4
FLOPs	64	400	4900	4900	64	784	4900	4900	64	576	5184	5184	64	400	2500	5625
ReLUs	64	80	140	70	64	112	140	70	64	96	144	72	64	80	100	75

D.5 Potential of ReLU Equalization as a Unified Network Design Principle

The manual network design such as ResNet (He et al., 2016), ResNeXt (Xie et al., 2017), ConvNeXt (Liu et al., 2022b), etc., and neural architecture search (Liu et al., 2018; Tan & Le, 2019; Howard et al., 2019; Tan et al., 2019) are the conventional way of designing state-of-the-arts neural networks for a wide range of complexity. While the former results in sub-optimal models when design choices increases, the latter lacks the interpretability and network design principles and fails to generalize beyond a restricted setting (Yang et al., 2020). In addition, both these network design paradigms require huge computation in order to find the optimal design hyper-parameters when networks are designed from scratch. On the other hand, the semi-automated design technique such as RegNets (Radosavovic et al., 2020) aims for interpretable network design and semi-automate the design procedure for finding the optimal population of networks generalizable across a wide range of settings. However, to gradually narrow down the search space, and access the quality of design space, it uses the error distribution function which requires a training sample size of thousands of models (Radosavovic et al., 2019) in each iteration. Consequently, it becomes expensive when models are designed from scratch.

In contrast, ReLU-equalization only needs the prior knowledge of stagewise criticality of baseline network — which are often same for a particular model family — thus needs to train very few models. Moreover, unlike the aforementioned design paradigms, it can be used for designing both the FLOPs and ReLU efficient neural networks. Note that the width and depth of RegNet networks are explained by a sophisticated quantized linear function, which eventually equalizes the networks' ReLU in their order of criticality. Thus, going forward, we believe that ReLU equalization will give a new perspective to simplify the network design for both FLOPs and ReLU efficient networks, and improve the interpretability.

D.6 HybReNet with Different Criticality Order

In this paper, we perform an exhaustive characterization of HRN networks designed for the prevalent criticality order Stage3 $>$ Stage2 $>$ Stage4 $>$ Stage1. However, we notice that criticality order of Stage2 and Stage4 get inter-changed in the following cases: (1) HRNs with α=2 originally designed for Stage3 $>$ Stage2 $>$ Stage4 $>$ Stage1 criticality order, and (2) ResNet18/ResNet34 on TinyImageNet (Jha et al., 2021). That is, in the aforementioned two cases criticality order changes to Stage3 $>$ Stage4 $>$ Stage2 $>$ Stage1. This brings a natural question, *do we need to run the criticality test for every baseline network on different datasets?* To answer this question, we need to compare the ReLU-accuracy performance of HRN networks designed with the aforementioned two different criticality orders. We apply the DeepReShape algorithm 1 for designing HybReNets for the given criticality order Stage3 $>$ Stage4 $>$ Stage2 $>$ Stage1 as follows.

$$\#ReLUs(S_3) > \#ReLUs(S_4) > \#ReLUs(S_2) > \#ReLUs(S_1)$$
$$\implies \phi_3\Big(\frac{\alpha\beta}{16}\Big) > \phi_4\Big(\frac{\alpha\beta\gamma}{64}\Big) > \phi_2\Big(\frac{\alpha}{4}\Big) > \phi_1$$

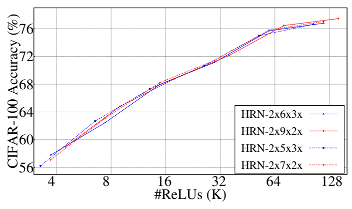
ReLU equalization through width $(\phi_1 = \phi_2 = \phi_3 = \phi_4 = 2, \text{ and } \alpha \geq 2, \beta \geq 2, \gamma \geq 2)$:

$$\implies \frac{\alpha\beta}{16} > \frac{\alpha\beta\gamma}{64} > \frac{\alpha}{4} > 1 \implies \alpha\beta > 16, \alpha > 4, \alpha\beta\gamma > 64, \beta > 4, \beta\gamma > 16, \text{ and } \gamma < 4$$

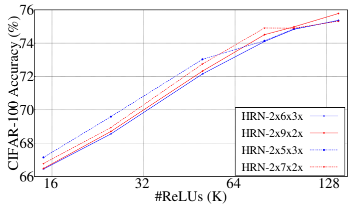
Solving the above compound inequalities provides the following range of β and γ at two different γ

$$\text{At } \gamma = 2,\ \beta > 8 \ \&\ \alpha > 4; \text{ and at } \gamma = 3,\ \beta > 5 \ \&\ \alpha > 4$$

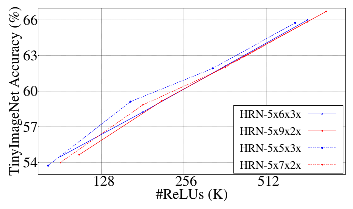
Thus, the HRNs with minimum values of α, β, and γ satisfying the ReLU equalization for the criticality order Stage3 $>$ Stage4 $>$ Stage2 $>$ Stage1 are HRN-5x6x3x and HRN-5x9x2x. Also, for achieving lower ReLU counts, HRN networks with α=2 is used, that is, HRN-2x6x3x and HRN-2x9x2x. We compare the ReLU-accuracy tradeoffs for these HRNs with the HRNs designed for prevalent criticality order, in Figure 14. With coarse-grained ReLU optimization, DeepReDuce, (Figure 14(a)) performance of HRNs for both the criticality order looks quite similar. We further employ the fine-grained ReLU-optimization, SNL, (Figure 14(b)) on the same set of networks, and observe a notice accuracy gap at lower ReLU counts. In particular, HRN-2x5x3x and HRN-2x7x2x outperform HRN-2x6x3x and HRN-2x9x2x by a small; however, noticeable margin. Further, we compare the performance of HRN-5x6x3x and HRN-5x9x2x on TinyImageNet in Figure 14(c). The performance of all the HRNs is quite similar, except at some intermediate ReLU counts HRN-5x5x3x outperforms by a noticeable margin.



(a) DeepReDuce on CIFAR-100



(b) SNL on CIFAR-100



(c) DeepReDuce on TinyImageNet

Figure 14: ReLU optimizations (DeepReDuce and SNL) on HRNs with different criticality orders.

Conclusively, even if we presume the default criticality order as the most prevalent one, Stage3 > Stage2 > Stage4 > Stage1, the designed HRN would not lose the performance compared to the HRNs designed for the exact same order of criticality.

E Experimental Results and Discussion for ReLU-reuse

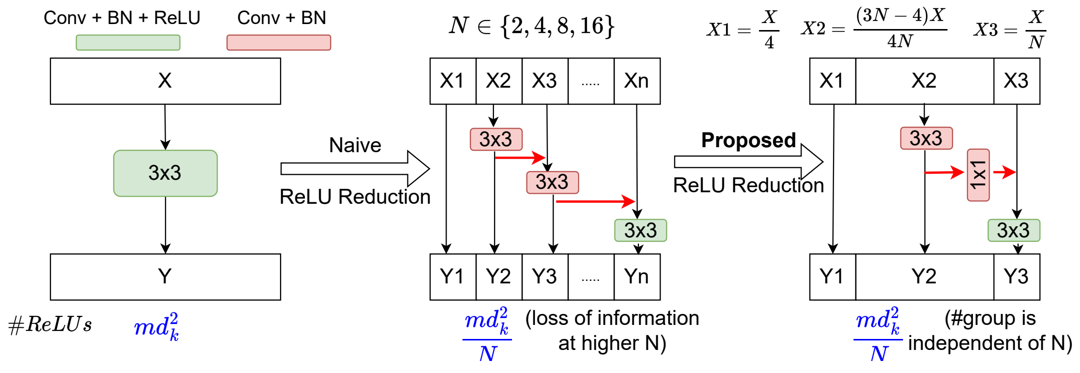

Conv + BN + ReLU
Conv + BN
X
3x3
Y
$\#ReLUs$ md_k^2
Naive
ReLU Reduction
$N \in \{2, 4, 8, 16\}$
X1 X2 X3 ..... Xn
3x3
Y1 Y2 Y3 ..... Yn
$\frac{md_k^2}{N}$ (loss of information at higher N)
$X1 = \frac{X}{4}$ $X2 = \frac{(3N-4)X}{4N}$ $X3 = \frac{X}{N}$
Proposed
ReLU Reduction
1x1
$\frac{md_k^2}{N}$ (#group is independent of N)

Figure 15: Proposed ReLU-reuse

Ablation study on ResNet18: We perform an ablation study on ResNet18 (CIFAR-100) and demonstrate the benefits of (1) having a shortcut connection between the output of one feature-subspace to the input of the next feature-subspace, and (2) having a fixed number of divisions, independent of the ReLU reduction factor. We employ the ReLU-reuse on the alternate layers on ResNet18, and the results are shown in Table 9. Initially, at lower ReLU reduction factors, there is a noticeable accuracy gain in the presence of shortcut connections. However, at higher ReLU reduction factors this accuracy gain decreases. In fact, in both the cases (with and without shortcut connections) accuracy drops by $\sim$**1.5%** for 4x reduction, compared to 2x reduction. In contrast, when we have a fixed number of divisions, accuracy drops at a higher ReLU reduction factor and remains (relatively)stable. Note that, at scale=2 the accuracy of our proposed ReLU-reuse is lower than that for the N division with shortcut connections. This is because, the latter has only two groups of feature maps while the former has three groups; thus, incurring more information loss.

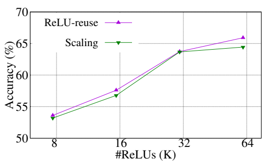
Table 9: We perform ablation study for ReLU-reuse on ResNet18 by applying them on alternate layers (CIFAR-100). For N divisions, with reuse denotes the presence of a shortcut connection between the output of a feature-subspace to the input of next feature-subspace. In our proposed ReLU-reuse, number of divisions is fixed, irrespective of the ReLU reduction factor.

ReLU reduction factor	ReLU count	N divisions		**Proposed**
		w/o Reuse	w/ Reuse	(3 divisions)
2x reduction (Scale=2)	434.18K	77.61%	78.19%	77.83%
4x reduction (Scale=4)	372.74K	75.84%	76.87%	77.60%
8x reduction (Scale=8)	342.02K	75.43%	75.66%	76.93%
16x reduction (Scale=16)	326.66K	75.33%	75.47%	76.38%

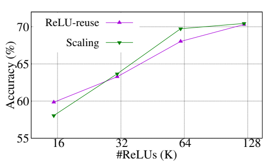
Performance comparison of HRN vs classical networks for ReLU-reuse: We now compare the performance of ReLU-reuse with the conventional (channel/feature-map) scaling used in DeepRe-

Duce (Jha et al., 2021) on both the classical networks and HRNs. First, we employ ReLU-reuse on all the layers of networks and reduce the ReLUs by a factor of $N \in \{2, 4, 8, 16\}$[2]. Results are shown in Figure 16. Evidently, for ResNet18 $BaseCh$ networks, the performance of ReLU-reuse is inferior to conventional scaling methods, and the performance gap increases for ResNet18 with higher base channel counts. In contrast, on the HRN networks, ReLU-reuse outperforms conventional scaling at lower ReLU counts; however, at higher ReLUs (especially for the ReLU reduction factor of two) the information loss incurring from the feature maps division dwarfs the benefit of ReLU-reuse. Note that, a similar observation is held for networks with partial ReLU-equalization, ResNet18(m=16)-4x4x4x.

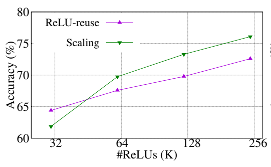
We further repeat the above experiments on Thinned networks as Reshaping is performed on Thinned networks in DeepReDuce (Jha et al., 2021). That is, first ReLUs are dropped from alternate layers and then ReLU-reuse is employed in the remaining layers. Results are shown in Figure 17. Since the ReLU-reuse is now employed in only one-half of the total number of layers, the cumulative information loss incurring from the loss of cross-channel information in feature-map divisions is also reduced. Consequently, the performance of ReLU-reuse is further boosted which is evident from the change in the performance gap between ReLU-reuse and conventional scaling for all the networks in Figure 17. In particular, compared to the results in Figure 16, the performance gap is reduced where ReLU-reuse was inferior (in ResNet18 $BaseCh$ network), and increases where ReLU-reuse was superior (in HRN networks).



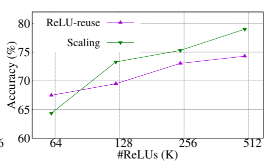
(a) ResNet18(m=16)



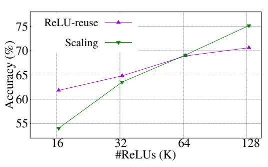
(b) ResNet18(m=32)



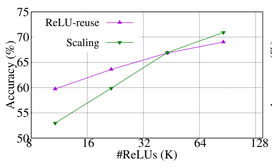
(c) ResNet18(m=64)



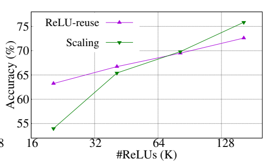
(d) ResNet18(m=128)



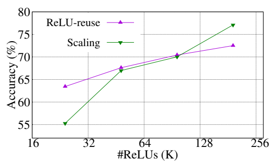
(e) 4x4x4x



(f) HRN-2x6x2x



(g) HRN-5x5x3x



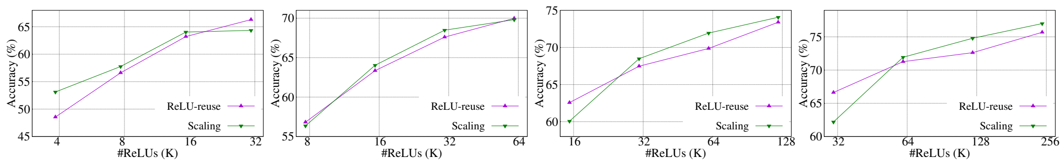
(h) HRN-6x6x2x

Figure 16: Performance comparison (CIFAR-100) of ReLU-reuse vs conventional scaling (used as reshaping steps in DeepReDuce Jha et al. (2021)) when ReLU-reuse is employed after every convolution layer.

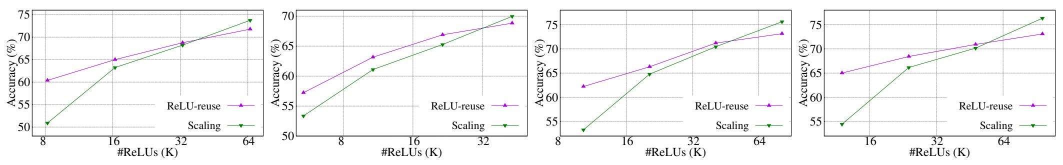
Incorporating ReLU-reuse in DeepReDuce pipeline: We incorporate the ReLU-reuse in the ReLU optimization pipeline of DeepReDuce, as shown in Algorithm 2.The original DeepReDuce runs for $(D-1)\times$ iterations since the input baseline was a classical network (e.g., ResNet18) where ReLUs' distribution was *agnostic* to their criticality order; thus, running a single iteration often results into *sup-optimal* performance. In contrast, DeepReDuce with ReLU-equalized HRN networks requires only one iteration as the ReLUs' distribution follows their criticality order. Thus, reduces the computational complexity of the original DeepReDuce by $(D-1)\times$. Note that, after Thinning, we use conventional channel scaling by reducing the ReLU count by $2\times$, rather than employing ReLU-reuse, as the latter performs inferior due to the cross-channel information loss (shown in Figure 16 and Figure 17).

Finally, we compare the results in Figure 18 for the HRN networks. Note that, here all the HRNs have α=2 as the lowering α values increase the proportion of (non-critical)Stage1 ReLUs, which are shown to be beneficial for achieving very low ReLU counts (see the results for criticality capacity tradeoff in Figure 8 and Figure 12)). We note an accuracy gain upto **3%**, at the iso-ReLU count, over the conventional scaling in the DeepReDuce. For instance, HRN-2x9x2x outperforms conventional scaling by 3.14% at 9K ReLUs (see Figure 12)(e)). The accuracy gain from the ReLU-reuse, helps achieve the performance at par with the fine-grained SNL at extremely low ReLU counts. Nonethe-

[2]For $N = 2$, we employ the naive ReLU reduction since it outperforms the proposed ReLU-reuse (see Figure 15)



(a) ResNet18(m=16) (b) ResNet18(m=32) (c) ResNet18(m=64) (d) ResNet18(m=128)



(e) 4x4x4x (f) HRN-2x6x2x (g) HRN-5x5x3x (h) HRN-6x6x2x

Figure 17: Performance comparison (CIFAR-100) of ReLU-reuse vs conventional scaling (used as reshaping steps in DeepReDuce Jha et al. (2021)) when ReLU-reuse is employed in Thinned networks. That is, first ReLU is dropped from every-alternate layers, and then ReLU-reuse is applied in remaining layers.

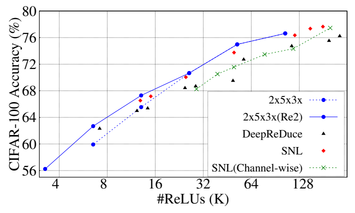
Algorithm 2 DeepReDuce for HRN networks with our proposed ReLU-reuse

Input: A network Net with D stages S_1, S_2, ..., S_D and C, a sorted list of stages from least to most critical

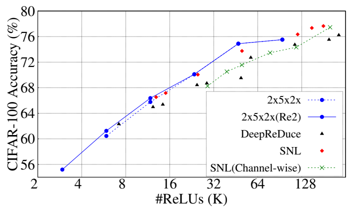
Output: ReLU optimized versions of Net

```
1: if the least critical stage C[1] has the highest fraction of ReLUs then
2:     $S_k$ = $C[1]$                                        ▷ Get the least critical stage
3:     $Net$ = $Net$ - $S_k$                                 ▷ Cull the least critical Stage $S_k$
4: end if
5: $Net_i^T$ = Thin($Net$)                                   ▷ Thin the remaining stages
6: $Net_i^C$ = ScaleCh($Net_i^T$, $\alpha$=0.5)              ▷ Channel scaled by 0.5x
7: $Net_i^{R4}$ = ReuseReLU ($Net_i^T$, Sc=4)                ▷ ReLU-reuse with scaling factor 4
8: $Net_i^{R8}$ = ReuseReLU ($Net_i^T$, Sc=8)                ▷ ReLU-reuse with scaling factor 8
9: $Net_i^{R16}$ = ReuseReLU ($Net_i^T$, Sc=16)              ▷ ReLU-reuse with scaling factor 16
10: $Nets$ += $Net$, $Net_i^T$, $Net_i^C$, $Net_i^{R4}$, $Net_i^{R8}$, $Net_i^{R16}$   ▷ Apply KD to each Net
11: return $Nets$
```

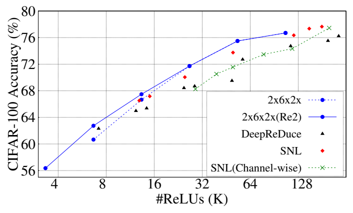
less, for fair comparison, we also plot the channel-wise SNL in Figure 12. Evidently, all the HRNs with ReLU-reuse achieves a substantial accuracy gain over channel-wise SNL.



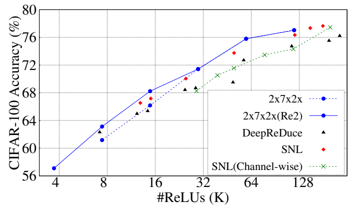
(a) HRN-2x5x3x



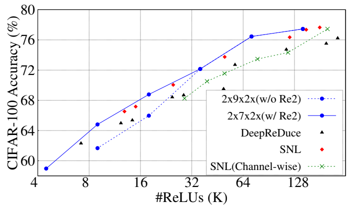
(b) HRN-2x5x2x



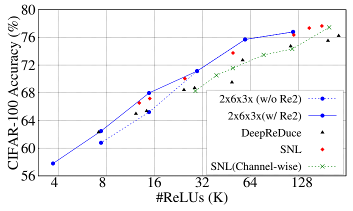
(c) HRN-2x6x2x



(d) HRN-2x7x2x



(e) HRN-2x9x2x



(f) HRN-2x6x3x

Figure 18: Experimental results for ReLU-reuse on CIFAR-100. We plot channelwise SNL, an unstructured ReLU dropping, for fair comparison with our proposed ReLU-reuse (denoted as Re2), a structured ReLU dropping. We note that with ReLU-reuse **a gain of 1% - 3% accuracy at iso-ReLU** has been achieved (on CIFAR-100). Compared to channel-wise SNL, HRNs achieves a substantial gain in accuracy on all the ReLU counts.

F Experimental Details

Sweeping the depth and width hyperparameters for ReLU-efficiency experiments To show the effect of depth in WideResNet models, we fixed the width and change the depth by sweeping the depth parameter $Q \in \{16, 22, 28, 34, 40\}$ in WRN-Qx2.Further, to show the effect of width we select two WideResNet models and sweep the width parameter $k \in \{2, 4, 6, 8, 10, 12\}$ in WRN-22xk and WRN-28xk while depth is fixed. Similarly, for the depth vs width experiments on ResNet models we lowered the base channel count in ResNet18 to m=16 as the vanilla ResNet20, ResNet32, and ResNet56 have {16, 32, 64} #channels in their successive stages while that in ResNet18 is {64, 128, 256, 512}. This enables a fair comparison among ResNet models. We widen the ResNet ($BaseCh$)models by performing a sweep m ∈{16,32,64,128,256}. Doubling base channel count doubles the number of channels throughout the network in all the layers as the channels in successive stages of ResNet get multiplied by a factor of two.

Training hyperparameters and procedure: For all the baseline training and DeepReDuce, on both the CIFAR-100 (Krizhevsky et al., 2010) and TinyImageNet (Le & Yang, 2015; Yao & Miller, 2015) datasets, we use an initial learning rate of 0.1, mini-batch size of 128, momentum of 0.9, and 0.0004 weight decay factor. We train networks using cosine annealing learning rate scheduler (Loshchilov & Hutter, 2016) for 200 epochs on both CIFAR-100 and TinyImageNet; however, we perform 10 additional epochs of warmup on TinyImageNet. We use Hinton's knowledge distillation (Hinton et al., 2015) and set the hyper-parameters, temperature, and relative weight to cross-entropy loss on hard targets as 4 and 0.9, respectively. For SNL, we train the baseline networks with the aforementioned methodology; however, for mask generation, fine-tuning, and knowledge distillation, we used their default implementation.