# Selective Network Linearization for Efficient Private Inference

**Minsu Cho** [1] **Ameya Joshi** [1] **Siddharth Garg** [1] **Brandon Reagen** [1] **Chinmay Hegde** [1]

## Abstract

Private inference (PI) enables inference directly on cryptographically secure data. While promising to address many privacy issues, it has seen limited use due to extreme runtimes. Unlike plaintext inference, where latency is dominated by FLOPs, in PI non-linear functions (namely ReLU) are the bottleneck. Thus, practical PI demands novel ReLU-aware optimizations. To reduce PI latency we propose a gradient-based algorithm that selectively linearizes ReLUs while maintaining prediction accuracy. We evaluate our algorithm on several standard PI benchmarks. The results demonstrate up to $4.25\%$ more accuracy (iso-ReLU count at 50K) or $2.2\times$ less latency (iso-accuracy at 70%) than the current state of the art and advance the Pareto frontier across the latency-accuracy space. To complement empirical results, we present a "no free lunch" theorem that sheds light on how and when network linearization is possible while maintaining prediction accuracy. Public code is available at `https://github.com/NYU-DICE-Lab/selective_network_linearization`.

## 1. Introduction

Cloud-based machine learning frameworks motivate the setting of private inference (PI). At a high level, the vision of private inference is to enable a user to (efficiently) perform inference of their data on a model owned by a cloud service provider. But both parties wish to preserve their privacy and both the user's data and the service provider's model are encrypted prior to inference using cryptographic techniques.

Curiously, the difficulty in realizing this vision is the nonlinear operations of a deep network. Private execution of the linear operations in a network using ciphertext can be made essentially as fast as normal (plaintext) evaluation of the same operations with secret sharing techniques and input independent preprocessing phases. However, to privately evaluate ReLUs in the network, some version of Yao's Garbled Circuits (GC) is necessary, resulting in proportionally high latency costs and storage overheads. In a nutshell: standard deep network architectures (such as ResNets) are ill-suited for efficient PI, since *they contain far too many ReLUs*. See (Mishra et al., 2020; Ghodsi et al., 2020; 2021; Jha et al., 2021) and a detailed discussion below in Section 2.

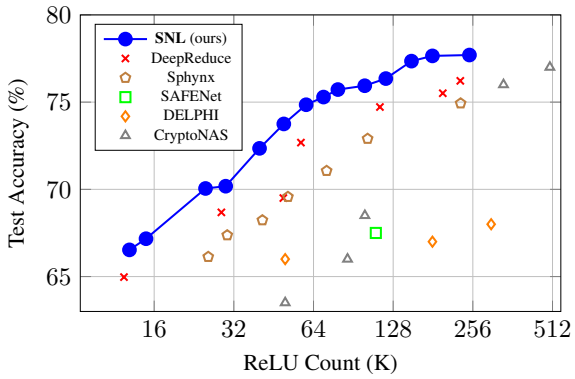



*Figure 1.* Our approach, SNL, achieves the Pareto frontier on ReLU counts vs. test accuracy for CIFAR-100.

Thus, unlocking the full potential of fast, accurate, and private neural inference require rethinking network design to have as few ReLUs as possible. Numerous efforts in this direction have already emerged. Prior work such as MiniONN (Liu et al., 2017) focus on the security protocols themselves, while more recent works such as Delphi (Mishra et al., 2020) or Circa (Ghodsi et al., 2021) propose to replace ReLUs with other activations. An alternative line of work designs ReLU-efficient network skeletons using neural architecture search (NAS). CryptoNAS (Ghodsi et al., 2020) uses evolutionary NAS, while Sphynx (Cho et al., 2021a) uses micro-search NAS.

Our work in this paper pursues a new path, and applies to several architectures that are used in imaging, computer vision, or other perception problems (specifically, deep convolutional networks with/without residual connections and ReLU activations). For these networks we study a problem that we call *deep network linearization*. This refers to the process of judiciously picking a subset of neurons in such a network and eliminating their nonlinearities (i.e., replacing

[1]New York University Tandon School of Engineering, New York. Correspondence to: Minsu Cho <mc8065@nyu.edu>.

*Proceedings of the $39^{th}$ International Conference on Machine Learning*, Baltimore, Maryland, USA, PMLR 162, 2022. 

their ReLU operation with an identity, or linear, operation) minimizing tradeoff between overall performance and number of ReLU operations. Note that via this process, we are *not* reducing the number of parameters in the network. Since ReLU activations are simple scalar operations, in normal scenarios we are essentially not getting any reduction in FLOP counts during network training or testing. However, in the PI setting, a major advantage of linearization is due to reducing the number of GC computations.

This paper introduces a simple gradient-based algorithm (that we call *Selective Network Linearization*, or SNL) to solve the deep network linearization problem, validates this algorithm on a variety of standard benchmarks, and provides theoretical evidence that illuminates its observed behavior.

Currently, the state-of-the-art in private inference is achieved by DeepReDuce (Jha et al., 2021). The authors introduce a concept known as *ReLU dropping* in deep networks that, at a conceptual level, is equivalent to network linearization. Their proposed algorithm consists of three stages: ReLU "culling", ReLU "thinning", and ReLU "reshaping", each of which requires several manual design choices, involving several hyper-parameters. The search space of candidate architectures is very large; applying DeepReDuce to a ResNet with $D$ stages would require training $\Omega(D)$ different networks and picking the one with the best accuracy. Also, crucially, the keep/drop decisions made by DeepReDuce are made in *stages*: either entire ReLU layers are linearized, or entire ReLU layers are retained as is.

In contrast, our proposed technique, SNL, is highly-automated, involving very few hyper-parameters. SNL is implemented using a single gradient-based training procedure (which we describe below in detail) and requires no searching over multiple network skeletons. Finally, SNL provides fine-grained control, all the way down to the pixel feature map level, of which ReLUs to retain or eliminate.

The intuition underlying SNL is simple and applicable to many deep networks. Consider any standard architecture (e.g., a ResNet-34), except with a twist. All ReLUs are now replaced with *parametric ReLUs* (or pReLUs) (He et al., 2015) with an *independent, trainable slope parameter* for *each ReLU* in the network. Further, each slope parameter is further constrained to be binary (0 or 1). Having defined this network, SNL proceeds to perform standard training (using, say, SGD or Adam); there is no other manual intervention necessary.

Some care is needed to make things work. The main challenges lie in (a) enforcing the binary constraints on the slopes of the pReLUs, and (b) ensuring that only a small number of ReLUs are retained, i.e., the vector of slope parameters is (anti) sparse. Mirroring the approach adopted by certain network pruning algorithms (Lee et al., 2019; Cho et al., 2021b), we resolve these difficulties by augmenting the standard train error loss with an $\ell_1$-penalty term defined over the slope coefficients, decaying the weight of this penalty on a schedule, and applying a final rounding step that binarizes the slopes. See Section 3.

We validate SNL on a variety of benchmark datasets commonly used in the literature on private image classification, and show that SNL *achieves Pareto dominance over the entire accuracy-latency tradeoff curve over all existing approaches*. Figure 1 above demonstrates this in the context of CIFAR-100; we provide several additional results (and ablation studies) below and also show the same benefits for CIFAR-10 and Tiny ImageNet. See Section 4 and Appendix.

Probing into the results of SNL reveals curious behavior. It appears that ReLUs in earlier layers are selectively linearized at a (much) higher rate than ReLUs in later layers; this aligns with the conclusions arrived at by earlier PI works such as DeepReDuce (Jha et al., 2021). Could this be indicative of some fundamental property of network learning? In a step towards rigorously answering this question, we prove *no free lunch* theorems showing that linearization comes at the cost of reducing memorization capacity of 3-layer networks. Moreover, if the network is *contractive*, i.e., if the second layer has fewer neurons than the first layer (typical in classification-type scenarios), then selective linearization retains original network capacity *only if* fewer neurons in the second layer are linearized. See Section 5.

## 2. Related Work

**Private inference.** Prior work on private inference (PI) have proposed methods that leverage existing cryptographic primitives for evaluating the output of deep networks. Cryptographic protocols can be categorized by choice of ciphertext computation used for linear and non-linear operations in a network. Operations are computed using some combination of: (1) secret-sharing (SS) under a secure multi-party computation (MPC) (Shamir, 1979; Goldreich et al., 2019) model; (2) partial homomorphic encryptions (PHE) (Gentry & Halevi, 2011), which allow limited ciphertext operations (e.g. additions and multiplications), and (3) garbled circuits (GC) (Yao, 1982; 1986) that rely on specialized circuitry.

In this paper, our focus is exclusively on the DELPHI protocol for private inference[1]. DELPHI assumes the threat model that both parties are honest-but-curious. Therefore, each party strictly follows the protocol, but may try to learn information about the other party's input based on the tran-

[1]We chose DELPHI as a matter of convenience; the general trends discovered in our work hold regardless of the encryption protocol. We acknowledge that choosing the "best" protocol is an important (and fast advancing) area of research, and that better PI protocols, such as (Rathee et al., 2020), may exist.

| Approach | Methods | Reduce ReLUs | Units that are removed |
|---|---|---|---|
| CryptoNAS (Ghodsi et al., 2020) | NAS | Yes | layers |
| Sphynx (Cho et al., 2021a) | NAS | Yes | layers |
| DELPHI (Mishra et al., 2020) | NAS + polynomial approx. | Yes | layers |
| SAFENet (Lou et al., 2021) | NAS + polynomial approx. | Yes | channels |
| Unstructured Pruning | N/A | No | not exist |
| Structured Pruning | N/A | Yes | channels, layers |
| DeepReDuce(Jha et al., 2021) | manual | Yes | layers |
| SNL (ours) | gradient-based | Yes | pixels, channels |

*Table 1.* Comparison of various techniques that reduce ReLU operations in deep networks. NAS stands for neural architecture search. "Pruning" techniques eliminate entire neurons. SNL, our proposed gradient-based network linearization method, achieves the accuracy-latency Pareto frontier in private inference.

scripts they receive from the protocol. This threat model is standard in prior PI works, including MiniONN (Liu et al., 2017), DELPHI, CryptoNAS (Ghodsi et al., 2020), SAFENet (Lou et al., 2021), and DeepReDuce (Jha et al., 2021).

DELPHI is a hybrid protocol that combines cryptographic primitives such as secret sharing (SS) and homomorphic encryptions (HE) for all linear operations, and garbled circuits (GC) for ReLU operations. DELPHI divides the inference into two phases to make the private inference happen: the offline phase and an online phase. DELPHI's cryptographic protocol allows for front-loading all input-independent computations to an offline phase. By doing so, this enables ciphertext linear computations to be as fast as plaintext linear computations while performing the actual inference.

For non-linear (especially ReLU) operations, all the above approaches leverage GC's for secure computation. However, such circuits cause major latency bottlenecks. Mishra et al. (2020) show empirical evidence that ReLU computation requires $90\%$ of the overall private inference time for typical deep networks. As a remedy, DELPHI and SAFENET (Lou et al., 2021) propose neural architecture search (NAS) to selectively replace ReLUs with polynomial operations. CryptoNAS (Ghodsi et al., 2020), Sphynx (Cho et al., 2021a) and DeepReDuce (Jha et al., 2021) design new ReLU efficient architectures by using macro-search NAS, micro-search NAS and multi-step optimization respectively.

**Network compression.** We emphasize that network linearization is *distinct* from neural network compression, which fundamentally focuses on reducing the number of *learnable parameters* in a deep network. Broadly, network compression (or pruning) techniques can be divided into two buckets. *Unstructured* pruning approaches identify and remove "unimportant" weights (edges) in a deep network to reduce model size. Zhu & Gupta (2017); Gale et al. (2019) retain top-$k$ parameters by measuring the importance of weight parameters based on their absolute magnitude. Lee et al. (2019) proposes measuring importance according to gradient magnitudes. Xiao et al. (2019); Cho et al. (2021b) trains the network with iterative multi-objective loss functions and reparameterized weights.

In contrast, *structured* pruning approaches remove entire channels in convolutional architectures; this has the effect of eliminating both nodes and edges in the network. Li et al. (2016) uses a combination of magnitude-pruning and fine-tuning to prune channels in every layer. He et al. (2020a) achieves state-of-the-art pruning performance in this direction. While structured pruning methods do end up reducing ReLU counts as a by-product, this reduction comes at the cost of losing expressivity. On the other hand, selective linearization retains the original number of learnable parameters while only reducing the number of ReLUs in the network. Below we show extensive experiments (and preliminary theory) showing why selective linearization is superior in the context of private inference.

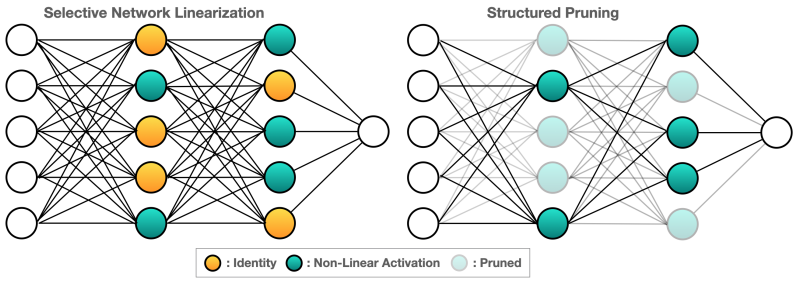
## 3. Selective Network Linearization

**Notation.** We represent matrices and tensors with uppercase variables unless stated otherwise. Elements of vectors and matrices are represented using appropriate subscripts. $\|\cdot\|_0$ represents number of non-zeros; $\odot$ represents element-wise products. $[d]$ is the natural number set $\{1, 2, 3, \ldots, d\}$. $\mathbf{1}$ is the all-ones vector of the appropriate dimension.

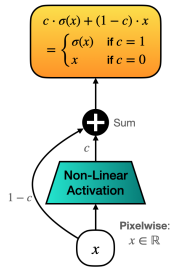
### 3.1. Setup

Our approach applies to any feed-forward deep network architecture with $d$ layers and ReLU activations ($\sigma(\cdot)$). Let $f_{\mathbf{W}}$ be any such network where $\mathbf{W} = \{W^0, W^1, \ldots, W^d\}$ are the layer weight. Denote the pre-activation outputs of each linear layer by $z^1, z^2, \ldots, z^d$. Denote $x$ be the input vector and $z^1 = W^1 x$. Let $a^i = \sigma(z^i)$ be the outputs after each layer of ReLUs. Expanding $f_{\mathbf{W}}(x)$, we get

$$f_{\mathbf{W}}(x) = W^d(\sigma(W^{d\ 1}(\sigma(W^{d\ 2}(\cdots\sigma(W^1 x)\cdots). \quad (1)$$




(a) Visual comparison on SNL and Structured Pruning




(b) Visual representation of Equation 2

*Figure 2.* Visualization of SNL, structured pruning, and Equation 2. (a) Both SNL and structured pruning have two non-linear activations. While all 55 parameters are on in SNL, the network from structured pruning has only 18 parameters. We note that number of non-linear activations (especially ReLU) is what matters in PI. (b) Visual representation of the convex combination between $x$ and $\sigma(x)$. If non-linear activation $\sigma$ is ReLU and $c$ 2 R, then this convex combination is equivalent to PReLU.

We now introduce a set of auxiliary binary vectors $\mathbf{C} = \{c^1, c^2, \dots c^d\}$, where for $i \in [d]$, we define $\sigma_{c^i}$ as the *parametric ReLU* (PReLU) activation function (with (left) slope equal to $1 - c^i$) defined for *every neuron* in the network. Then, the output, $\sigma_{c^i}(\cdot)$, at the $i^{\text{th}}$ layer can be formally expressed as:

$$a^i_{c^i} = \sigma_{c^i}(z^i) = c^i \odot \sigma(z^i) + (\mathbf{1} - c^i) \odot z^i \qquad (2)$$

Notice that according to our definition, $a^i_{c^i} = \text{ReLU}(z^i)$ for $c_i = 1$, and $a^i_{c^i} = z^i$ for $c_i = 0$. In essence, we are re-defining $\sigma$ to $\sigma_{c^i}$ for all $i \in [d]$ in equation 1, so that now the feed-forward deep network $f_{\mathbf{W},\mathbf{C}}$ is parameterized both by $\mathbf{W}$ and $\mathbf{C}$.

Our goal is to learn a model with high accuracy but with as few ReLUs as possible. This motivates the following $\ell_0$-constrained optimization problem:

$$\min_{\mathbf{W},\mathbf{C}} L(f_{\mathbf{W},\mathbf{C}}(\mathbf{x}), \mathbf{y}) \quad \text{s.t.} \sum_{i=1}^{d} \|c_i\|_0 \leq B,\ c_i \text{ binary}. \qquad (3)$$

In terms of optimization difficulty, the main challenges with Equation 3 are in enforcing the binary constraints on each entry of $c_i$, and the overall $\ell_0$ constraint. Below, we propose a solution to these challenges.

### 3.2. The SNL Algorithm

We first drop the binary constraints, and relax the optimization problem in Equation 3 to a Lasso-style objective function (Tibshirani, 1996):

$$\min_{\mathbf{W},\mathbf{C}} L(f_{\mathbf{W},\mathbf{C}}(\mathbf{x}), \mathbf{y}) + \lambda \left( \sum_{i=1}^{d} \|c_i\|_1 \right). \qquad (4)$$

In principle, both the weights $\mathbf{W}$ and the slope parameters $\mathbf{C}$ can be updated via standard gradient iterations over this objective. In practice, to ensure good performance, our approach requires four steps: (1) Start with a pre-trained network $f_{\mathbf{W}}$. (2) Update $\mathbf{W}$ and $\mathbf{C}$ simultaneously by gradient descent over Equation 4. (3) Binarize elements of $\mathbf{C}$ by rounding to 0 or 1. (4) freeze $\mathbf{C}$ and perform final finetuning of $f_{\mathbf{W}}$.

**Algorithm 1** SNL: Selective Network Linearization

**Inputs:** $f_{\mathbf{W}}$: pre-trained network, $\lambda$: Lasso coefficient, $\kappa$: scheduling factor, $B$: ReLU budget, $\epsilon$: threshold.
Set $\mathbf{C}$ = 1: same dimensions to all feature maps.
$\overline{\mathbf{W}}$ $(\mathbf{W}, \mathbf{C})$
**while** ReLU Count $> B$ **do**
  Update $\overline{\mathbf{W}}$ via ADAM for one epoch.
  ReLU Count $k\mathbb{1}(\mathbf{C} > \epsilon)k_0$
  **if** ReLU count not decreased **then**
    Increment Lasso coefficient $\lambda$ $\kappa$ $\lambda$.
  **end if**
**end while**
$\mathbf{C}$ $\mathbb{1}(\mathbf{C} > \epsilon)$
Freeze $\mathbf{C}$ and finetune $f_{\mathbf{W}}$.

In Step 2, we first initialize $\mathbf{C}$ to all ones in the vector space. Then, we update $\mathbf{W}$ and $\mathbf{C}$ simultaneously via standard backpropagation (using ADAM), until the desired sparsity level of $\mathbf{C}$ is achieved. We monitor the sparsity of $\mathbf{C}$ by computing the quantity $\|\mathbb{1}(\mathbf{C} > \epsilon)\|_0$, where $\epsilon$ is a very small non-negative hyperparameter. Similar to the technique of *homotopy continuation* in Lasso, if the sparsity of $\mathbf{C}$ increases compared to previous epochs we increase $\lambda$ with a small multiplicative factor $\kappa$ and repeat. Once the desired number of nonzero elements in $\mathbf{C}$ reached, we binarize by thresholding $\mathbb{1}(\mathbf{C} > \epsilon)$. Finally, we freeze $\mathbf{C}$ and finetune the network weights $\mathbf{W}$ to boost final performance. Algorithm 1 provides detailed pseudocode.

### 3.3. Aside: Linearization versus pruning

Let us pause to highlight the essential diffences of Equation 3 with unstructured network pruning (Zhu & Gupta, 2017; Lee et al., 2019; Cho et al., 2021b). There, one typi-

cally chooses a *masking* parameter for every *weight* in the network, and solves a similar looking optimization:

$$\min_{\mathbf{w},\mathbf{m}} L(f(\mathbf{w} \odot \mathbf{m}; \mathbf{x}), \mathbf{y}) \quad \text{s.t. } \|\mathbf{m}\|_0 \leq k \tag{5}$$

where $k$ is a parameter counting the total number of retained weights.

But eliminating (individual) edges do not lead to reduction in ReLU counts; only if *all* incoming weights to a given ReLU operation are removed – or if an entire channel is removed – then one can safely eliminate the corresponding ReLU operation in the feature map. This is akin to *structured* pruning, but as we will show below, this leads to significantly worse performance as well as representation capacity compared to SNL. Figure 2(a) visualizes the difference between SNL and structured pruning.

## 4. Experiments

**Architectures, datasets, training.** We apply SNL to the ResNet18/34 architectures(He et al., 2016) and Wide-ResNet 22-8 (Zagoruyko & Komodakis, 2016) architectures. As standard in prior work (Jha et al., 2021), we remove the ReLU layers in the first convolution layer. This layer serves to raise the input channel dimension from 3 to a higher number (e.g. 64 for ResNet18/34).

We focus on standard image classification datasets (CIFAR-10/100 and Tiny ImageNet). The image resolutions on both CIFAR-10 and CIFAR-100 is $32 \times 32$; Tiny-ImageNet is $64 \times 64$. CIFAR10 has 10 output classes with $5000$ training images and $1000$ test images per class, while CIFAR-100 has 100 output classes with $500$ training images and $100$ test images per class. TinyImageNet has 200 output classes with 500 training images and 50 validation images. .

We first pre-train networks on CIFAR-10/100 using SGD with initial learning rate 0.1 and momentum 0.9, decay the learning rate at 80 and 120 epochs with 0.1 learning rate decay factor, 0.0005 weight decay, and use batch-size 256. For Tiny-ImageNet, we use the same hyperparameters, except that we train for 100 epochs and perform learning rate decay at 50 and 75 epochs with decay factor 0.1.

For the SNL algorithm, we initialize $\lambda = 0.00001$ and increment $\lambda$ by a multiplicative factor $1.1$ if the ReLU count increases compared to previous epochs. We set the thresholding parameter $\epsilon = 0.01$. We use the ADAM optimizer with learning rate equual to $0.001$. In the finetuning stage, we update the network parameter $W$ via SGD with a learning rate of $0.001$ and 0.9 momentum for 100 epochs. Finally, like DeepReDuce, we use knowledge distillation during finetuning (Hinton et al., 2015) with temperature parameter $4$ and equal relative weights on both cross-entropy (on the hard labels) and KL divergence (on the soft labels). We use the original pretrained model with identical network topology as the teacher.

**Latency estimates for private inference.** We report results both in terms of ReLU counts and wall-clock time for private inference. Existing open-source implementations of DELPHI are unfortunately not compatible with networks with pixel-wise parametric ReLU operations. Therefore, we propose a method to empirically estimate the online latency for private inference over such networks per input data sample. We break down the sources of latency into two categories: ciphertext linear operations and ciphertext ReLU operations. Since ciphertext linear operations in DELPHI are essentially as fast as plaintext linear operations (and since ReLUs in plaintext are essentially free), we simply report the latency of ciphertext linear operations as the same as that for one plaintext network inference. For ciphertext ReLU operations, we experimentally measure the wall-clock time for $1000$ ReLU operations using DELPHI, resulting in $t = 0.021$ seconds per $1000$ ReLUs [2].

**Pareto analysis and comparisons.** Our main result, Figure 3, shows that SNL achieves the ReLU count-accuracy Pareto frontier on CIFAR-10, CIFAR-100, as well as Tiny-ImageNet over all previous competing approaches.

To our knowledge, the previous best approach is due to DeepReDuce, which (as we discussed in the Introduction) optimizes networks in three stages, each with a fair bit of manual intervention. On the other hand, SNL outperforms DeepReDuce with a fairly straightforward gradient-based procedure. Furthermore, we used identical hyperparameters for all target ReLU budgets and base networks, showing that SNL is capable of giving robust state-of-the-art performance without careful hyperpararameter tuning.

We now discuss our results in detail in two regimes of interest: high-ReLU and low-ReLU budgets (Table 2). At a high-ReLU budget, SNL achieves 76.35% test accuracy with budget=120K on CIFAR-100, while DeepReDuce achieves 75.50% accuracy given a significantly higher (197K) ReLU count. Therefore, the network produced by SNL cuts down latency to $60\%$ of that of DeepReDuce, while achieving $0.85\%$ higher accuracy. In comparison, the network from CryptoNAS with a comparable performance as SNL requires nearly *3 times* the total ReLU counts. Similarly, the network produced by Sphynx (Cho et al., 2021a) with achieves $74.93\%$ with ReLU count=230K; SNL matches this performance with approximately half the ReLU count.

[2] We verify our estimates with independent prior work. For example, Table 4 of DeepReDuce (Jha et al., 2021) lists number of ReLUs and latency for several models. By performing a linear regression between these covariates and measuring the slope, we can get an estimate of the online latency per ReLU as $t = 0.019$ seconds, which is almost exactly what we get.

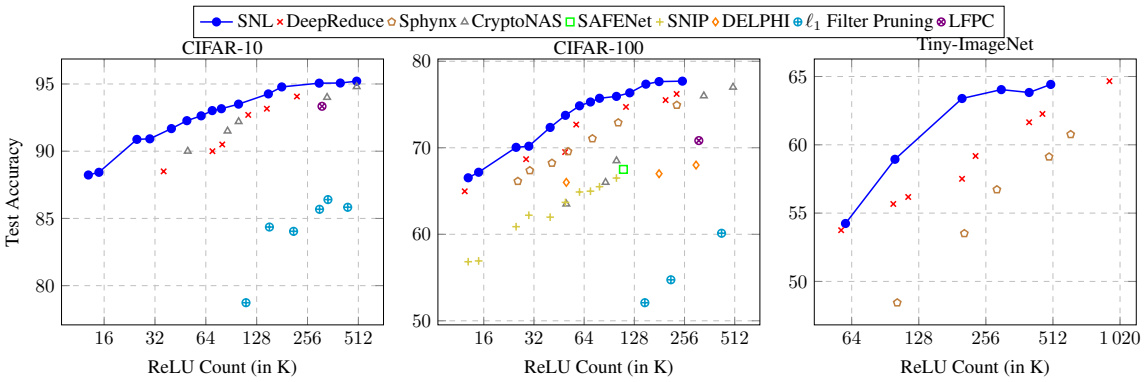



*Figure 3.* SNL achieves Pareto frontiers of ReLU counts versus test accuracy on CIFAR-10, CIFAR-100, and Tiny-ImageNet. SNL outperforms the state-of-the-art methods (DeepReDuce, SAFENet, and CryptoNAS) in all range of ReLU counts on all three dataset. $\ell_1$ Filter Pruning (Li et al., 2016) and LFPC (He et al., 2020b) are structured pruning techniques.

On Tiny-ImageNet, SNL produces a network that only uses $55\%$ of ReLUs as that of DeepReduce (500K vs. 917K) with par accuracy (64.42% vs. 64.66%).

At low-ReLU budgets, the SNL network achieves 73.75% test accuracy with 49.4K ReLU count on CIFAR-100. Both DeepReDuce and Sphynx use around 50K ReLU count, and achieve 69.50% and 69.57% test accuracy respectively. CryptoNAS network with 100K ReLU count reaches 68.75% test accuracy, but SNL beats CryptoNAS by nearly 5% in terms of accuracy with only half its ReLU budget. At the extreme case, we see that the SNL network with ReLU count 25K achieves $70.05\%$, outperforming DeepReDuce models with 68.68% containing ReLU count 28.6K. On Tiny-ImageNet, SNL achieves 63.39% with ReLU count 200K, whereas DeepReDuce network reaches 61.65% accuracy with nearly double the ReLU count (400K). Furthermore, Sphynx achieves 4% less accuracy with $3\times$ the ReLU counts (614K), compared to SNL with ReLU count 200K.

**Analysis of networks produced by SNL.** In SNL we only impose a total ReLU budget constraint; the algorithm automatically decides how it wants to allocate its budget across different layers. To study the effects of this allocation, we run SNL on ResNet-18 trained on CIFAR-100 with various ReLU budgets ranging from 20K to 300K; the base ResNet-18 requires approximately 492K ReLU operations. Comparing the blue and the orange bars in Figure 4, we see that SNL drops earlier ReLUs at a higher rate than later ReLUs; this indicates that ReLU operations on later layers are generally more crucial than the earlier layers. Indeed, the ReLU distribution plots with ReLU budget 100K, 200K, and 300K show that the ReLU operations in later layers (13, 14, 15, and 16) are mostly preserved. In contrast, the ReLUs in the earlier layers (1, 2, 3, 4) are largely eliminated and replaced with the identity operation. This observation mirrors the approach in DeepReDuce. There, the authors discovered by exhaustive manual ablation studies that ReLUs in later layers had a more significant effect than earlier layers, and therefore proposed a "ReLU criticality metric" to decide whether or not to cull entire layers of ReLUs. Interestingly, SNL arrives at a similar strategy without any manual intervention[3]. We revisit this in Section 5.

**Channel-wise SNL.** As described, the SNL algorithm proposes linearizing ReLUs at the finest (pixel-level) scale. We can develop "channel-wise" extensions of this method with a straightforward change-of-variables. Consider a CNN architecture, where our decision to linearize is at the channel (filter) scale. In this case, everything is the same as before, except that $\mathbf{C}$ in Algorithm 1 is the same dimension as the total number of convolution filters. Let $x_i^d$ be the feature map after applying the $i^{\text{th}}$ filter in the $d^{\text{th}}$ layers. We can selectively linearize this via:

$$z_i^d = c_i^d \cdot ReLU(x_i^d) + (1 - c_i^d) \cdot (x_i^d) \tag{6}$$

and the rest of the algorithm proceeds identically. We call this *Structured SNL* and test this with ResNet-18 and Wide-ResNet 22-8 on the CIFAR-100 dataset. Table 6 in the Appendix shows that structured SNL outperforms SAFENet with 0.7% higher accuracy (68.26% vs. 67.50%), with $11.5\times$ faster online inference time (0.628s vs. 7.20s).

**SNL outperforms structured pruning.** We also compare SNL to magnitude-based structured pruning. As discussed previously, structured pruning methods remove entire filters (and as a by-product also reduce ReLU counts). However, the loss of learnable parameters leads to significant losses in accuracy. We compare SNL, $\ell_1$-magnitude filter pruning (Li

[3] It appears that the authors of DeepReDuce measured criticality in ResNet *stages* and not individual layers. They found that the order of ResNet18 stages on CIFAR-100 as $S_1 < S_4 < S_2 < S_3$, where $S_i$ means a group of residual blocks (e.g., $S_3$ includes 9, 10, 11, and $12^{\text{th}}$ layers in ResNet-18)

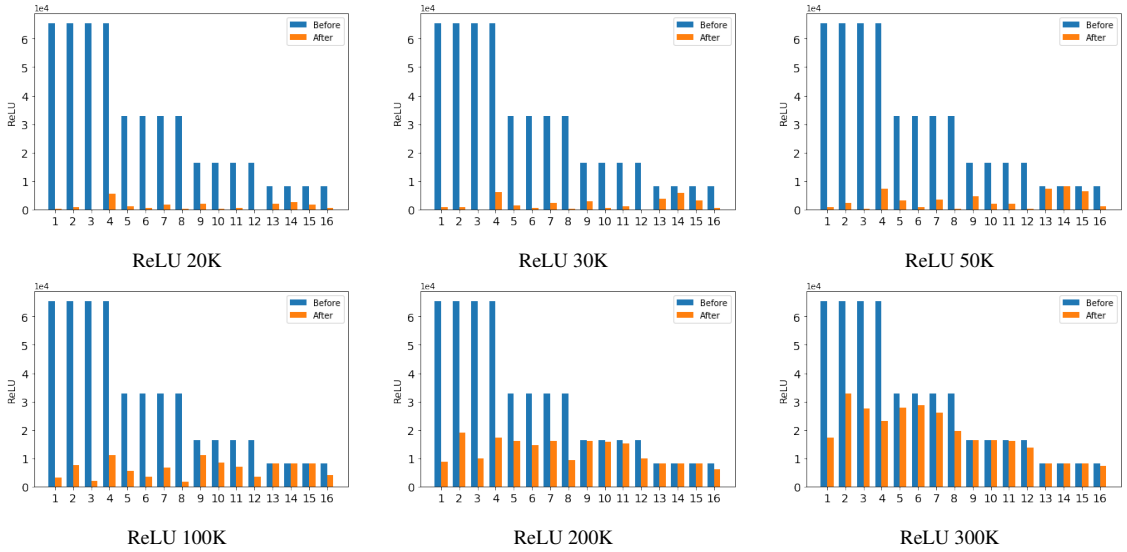



*Figure 4.* ReLU distribution on ResNet18 from SNL. *Before* and *After* stand for ReLU counts before SNL and after SNL, respectively. $x$-axis correpsonds to a layer index of the network. The larger gap between a blue and orange bar means the more ReLU operations replaced with identity function. Aligning to the theoretical results from Corollary 2, SNL replaces more ReLU operations to identity function in the earlier layers (1-4) while later layers' ReLU operations preserved (13-16).

*Table 2.* CIFAR-100 Comparison

| | Methods | #ReLUs (K) | Test Acc. (%) | Online Lat. (s) | Acc./ReLU |
|---|---|---|---|---|---|
| ReLU 100K | SNL[#] | 12.9 | 66.53 | 0.291 | 5.517 |
| | SNL[#] | 15.0 | 67.17 | 0.334 | 4.478 |
| | SNL[#] | 24.9 | 70.05 | 0.542 | 2.813 |
| | SNL[#] | 49.9 | 73.75 | 1.066 | 1.478 |
| | DeepReDuce | 12.3 | 64.97 | 0.45 | 5.282 |
| | DeepReDuce | 28.7 | 68.68 | 0.56 | 2.393 |
| | DeepReDuce | 49.2 | 69.50 | 1.19 | 1.413 |
| | Sphynx | 25.6 | 66.13 | 0.727 | 2.583 |
| | Sphynx | 51.2 | 69.57 | 1.335 | 1.359 |
| | Sphynx | 102.4 | 72.90 | 2.385 | 0.712 |
| | CryptoNAS | 100.0 | 68.5 | 2.30 | 0.685 |
| ReLU 300K | SNL[*] | 120.0 | 76.35 | 2.802 | 0.636 |
| | SNL[*] | 150.0 | 77.35 | 3.398 | 0.516 |
| | SNL[*] | 180.0 | 77.65 | 4.054 | 0.431 |
| | DeepReDuce | 197.0 | 75.51 | 3.94 | 0.383 |
| | DeepReDuce | 229.4 | 76.22 | 4.61 | 0.332 |
| | CryptoNAS | 344.0 | 76.0 | 7.50 | 0.221 |
| | Sphynx | 230.0 | 74.93 | 5.12 | 0.326 |

[#] Starts with pretrained ResNet18.
[*] Starts with pretrained Wide-ResNet 22-8.

et al., 2016) and LFPS-filter pruning (He et al., 2019) with pretrained Resnet-18 networks on CIFAR-10 and CIFAR-100. The results are available in Fig. 3. SNL outperforms $\ell_1$ pruning by a large margin on accuracy for similar ReLU counts. While LFPS-pruning is more accurate than $\ell_1$ pruning, SNL is able to achieve the same accuracy while only requiring $\sim 4\times$ fewer ReLUs. See also Appendix B for several additional ablation studies.

## 5. Theoretical analysis

Figure 4 in Section 4 reveals interesting trends. The blue bars indicate the number of ReLUs in every layer of a (standard) ResNet-18 model. Since the network is contractive, this number generally decreases as a function of depth. But after applying SNL, the red bars indicate that across the entire range of ReLU target budgets, *neurons in earlier layers are linearized at a far greater rate than later layers*. This aligns well with earlier findings; DeepReDuce (Jha et al., 2021) also hypothesized that (manually) culling all ReLUs in the first few layers provides the largest latency improvement with the smallest accuracy drop.

We now present theoretical evidence as to why this might be the case. One way to measure the effect of linearizing ReLUs in a network is to compare the *expressivity* of networks before and after linearization. To measure the expressivity, we leverage the notion of *memorization capacity*.

**Definition 1** (Yun et al. (2019); Vershynin (2020))**.** Let $f_W$ be a neural network of a given architecture with learnable parameters $W$. The memorization capacity of $f_W$ is the largest $N$ such that the following condition is satisfied: For all inputs $\{x_i\}_{i=1}^N \subset \mathsf{R}^{d_x}$ and for all $\{y_i\}_{i=1}^N \subset [-1, 1]$, there exists a choice of parameters $W$ such that $f_W(x_i) = y_i$ for $1 \leq i \leq N$.

Our main results focus on dense feed-forward neural networks (FNN) [4]. Under the assumption that the dataset

[4] Analogous extensions to convolutional networks with/without residual connections are interesting directions for future work.

$\{(x_i, y_i)\}_{i=1}^N$ contains distinct data points, (Yun et al., 2019) provide an upper bound on memorization capacity of FNNs with 2 hidden layers.

**Theorem 1** (Yun et al. (2019))**.** Consider an FNN $f_W$ with two hidden layers and piecewise linear activation function $\sigma$. Suppose that the hidden layers have width $d_1$ and $d_2$ and that the number of pieces in the activation function is $p$. Then, the memorization capacity of $f_W$ is no greater than:

$$p(p-1)d_1d_2 + (p-1)d_2 + 2\,.$$

For the case of ReLU networks, $p = 2$ and the above bound on the capacity simplifies to $2d_1d_2 + d_2 + 2$.

The intuition for this theorem is based on the fact that FNNs with ReLU activations implement piecewise affine functions; so an upper bound on their capacity can be obtained by counting the number of "linear pieces" in the range of the FNN, which can be recursively computed for each layer. See the appendix of (Yun et al., 2019) for a detailed proof.

Let us now imagine selectively linearizing such an FNN; more specifically, suppose we only linearize all but a fraction $\alpha_1$ of the neurons in the first layer, and all but a fraction $\alpha_2$ of the neurons in the second layer. Mirroring the same counting argument as above, we obtain the following corollary to Theorem 1.

**Corollary 1.** Consider an FNN $f_W$ with 2 hidden layers of width $d_1$ and $d_2$ respectively, and piecewise linear activation function $\sigma$ with $p$ pieces. Let $\alpha_i \in [0, 1], i \in \{1, 2\}$ be the fraction of nonlinear activations retained at each layer, i.e., without loss of generality the $l^{\text{th}}$ layer $k^{\text{th}}$ neuron exhibits the activation:

$$a_k^l(x) = \begin{cases} \sigma(z_k^l(x)) & \text{if } k \in [\alpha_l d_l] \\ z_k^l(x) & \text{if } k \in [d_l] \setminus [\alpha_l d_l] \end{cases}$$

Then, the memorization capacity of $f_W$ is no greater than:

$$\alpha_1\alpha_2 d_1 d_2 p(p-1) + \alpha_2 d_2(p-1) + \\ \alpha_1(1-\alpha_2)d_1d_2(p-1) + 2\,.$$

For ReLU networks, the above bound simplifies to: $\alpha_1(1 + \alpha_2)d_1d_2 + \alpha_2d_2 + 2$.

Corollary 1 shows that the memorization capacity of a selectively linearized network (obtained, say, by applying SNL) is strictly smaller than the original network whenever $\alpha_1$ or $\alpha_2$ is smaller than 1.

It is useful to see how selective linearization compares with (structured) pruning. Suppose that instead of just dropping the ReLUs activation, we eliminate the entire neuron from the network, i.e., the network widths now become $\alpha_1 d_1$ and $\alpha_2 d_2$ respectively. The following corollary can be obtained by directly instantiating these in Theorem 1.

**Corollary 2.** Consider a 3-layer FNN $f_W$ with piecewise linear activation $\sigma$ with $p$ pieces. Let $\alpha_1, \alpha_2 \in [0, 1]$ be the fraction of nonlinear activations retained at the first and second layers, respectively. Then, the memorization capacity of $f_W$ is no greater than:

$$\alpha_1\alpha_2 d_1 d_2 p(p-1) + \alpha_2 d_2(p-1) + 2 < N.$$

For ReLU networks, the above bound simplifies to $2\alpha_1\alpha_2 d_1 d_2 + \alpha_2 d_2 + 2$.

Notice the gap between the capacity upper bounds in Corollary 1 and Corollary 2. Specifically, selective linearization enjoys an additional additive term:

$$\alpha_1(1-\alpha_2)d_1d_2,$$

compared to pruning. In fact, equality between the capacity bounds is achieved when $\alpha_2 = 1$, i.e., when all the neurons in the second layer are retained.

This observation might indicate that retaining most ReLUs (i.e., driving $\alpha_2 \to 1$ in the second layer is the key. In fact, we can go further. From Corollary 1, if we fix an overall ReLU budget for the network, then we can derive optimal conditions on the fraction parameters $\alpha_1$ and $\alpha_2$ to maximize memorization capacity. To simplify notation, we restrict our attention to ReLU activations (i.e., $p = 2$).

**Theorem 2.** Consider a 3-layer FNN with ReLU activation. Let $\alpha_1, \alpha_2 \in [0, 1]$ be the fraction of nonlinear activations retaiend at the first and second hidden layers, respectively. Given a ReLU budget $B > d_2$, the choice $\alpha_1 = \frac{B+d_2}{2d_1} - \frac{1}{2}$ and $\alpha_2 = \frac{B}{2d_2} - \frac{d_2+1}{2}$ maximizes the memorization capacity.

This result confirms that the optimal "retention ratio" for each network is inversely proportional to its width, and therefore in contractive networks (such as those typical in classification) where $d_1 > d_2$, linearizing a greater number of neurons in the first layer is beneficial. As a concrete example, consider a 3-layer FNN with $d_1 = 50000$ and $d_2 = 5000$, and budget $B = 10000$. Then, the optimal choices are $\alpha_1 = 14999/100000 \approx 0.15$ and $\alpha_2 = 5001/10000 \approx 0.5$. Therefore, linearizing $42,500$ neurons in the first layer, and only $7,500$ neurons with ReLU in the second layer gives the best memorization capacity among all possible choices.

## 6. Discussion

We have developed a simple method to selectively linearize deep neural networks with ReLU activations, and using this method, demonstrated state-of-the-art networks for the problem of private inference.

Numerous directions for further work remain. Our work focused on reducing online runtime of private inference; a more holistic approach would consider both pre-processing

and online costs. Instead of selectively linearizing certain ReLUs, replacing with other activations may be beneficial. Finally, deriving tighter bounds on memorization capacity for general architectures appears doable.

## Acknowledgements

This work was supported in part by the National Science Foundation (under grants CCF-2005804 and 1801495), USDA/NIFA (under grant 2021-67021-35329), the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by Semiconductor Research Consortium (SRC) and the Defense Advanced Research Projects Agency (DARPA).

## References

Cho, M., Ghodsi, Z., Reagen, B., Garg, S., and Hegde, C. Sphynx: Relu-efficient network design for private inference. *arXiv preprint arXiv:2106.11755*, 2021a.

Cho, M., Joshi, A., and Hegde, C. Espn: Extremely sparse pruned networks. In *2021 IEEE Data Science and Learning Workshop (DSLW)*, pp. 1–8. IEEE, 2021b.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gentry, C. and Halevi, S. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 129–148. Springer, 2011.

Ghodsi, Z., Veldanda, A. K., Reagen, B., and Garg, S. Cryptonas: Private inference on a relu budget. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2020.

Ghodsi, Z., Jha, N. K., Reagen, B., and Garg, S. Circa: Stochastic relus for private deep learning. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2021.

Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 307–328. 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE Intl. Conf. Comp. Vision*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Intl. Conf. Comp. Vision*, pp. 770–778, 2016.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. *IEEE Conf. Comp. Vision and Pattern Recog*, pp. 4335–4344, 2019.

He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., and Yang, Y. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *IEEE Conf. Comp. Vision and Pattern Recog*, 2020a.

He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., and Yang, Y. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *IEEE Conf. Comp. Vision and Pattern Recog*, pp. 2009–2018, 2020b.

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

Jha, N. K., Ghodsi, Z., Garg, S., and Reagen, B. DeepReDuce: Relu reduction for fast private inference. In *Proc. Int. Conf. Machine Learning*, 2021.

Lee, N., Ajanthan, T., and Torr, P. H. S. Snip: Single-shot network pruning based on connection sensitivity. In *Proc. Int. Conf. Learning Representations*, volume abs/1810.02340, 2019.

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

Liu, J., Juuti, M., Lu, Y., and Asokan, N. Oblivious neural network predictions via minionn transformations. In *Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security*, pp. 619–631, 2017.

Lou, Q., Shen, Y., Jin, H., and Jiang, L. Safenet: Asecure, accurate and fast neu-ral network inference. In *Proc. Int. Conf. Learning Representations*, 2021.

Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., and Popa, R. A. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. USENIX Association, Aug. 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., and Sharma, R. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 325–342, 2020.

Shamir, A. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Telgarsky, M. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

Vershynin, R. Memory capacity of neural networks with threshold and rectified linear unit activations. *SIAM Journal on Mathematics of Data Science*, 2(4):1004–1033, 2020.

Xiao, X., Wang, Z., and Rajasekaran, S. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

Yao, A. C.-C. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 160–164. IEEE, 1982.

Yao, A. C.-C. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 162–167. IEEE, 1986.

Yun, C., Sra, S., and Jadbabaie, A. Small relu networks are powerful memorizers: a tight analysis of memorization capacity. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Adv. Neural Inf. Proc. Sys. (NeurIPS)*, 2019.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

# A. Proofs

## A.1. Proof of Corollary 1

*Proof.* The proof is based on the idea of counting the number of pieces in the range of networks with piecewise linear activations. Consider any vector $u \in \mathsf{R}^{d_x}$, and define the following dataset: $x_i = iu, y_i = (-1)^i$, for all $i \in [N]$. From (Telgarsky, 2015; Yun et al., 2019) we have:

**Lemma 1.** If $g : \mathsf{R} \to \mathsf{R}$ and $h : \mathsf{R} \to \mathsf{R}$ are piecewise linear with $k$ and $l$ linear pieces, respectively, then $g + h$ is piecewise linear with at most $k + l - 1$ pieces, and $g \circ h$ is piecewise linear with at most $kl$ pieces.

Fix $u$ and consider the output of layer 1 for input $x = tu$, where $t$ is a scalar variable: $\bar{a}^1(t) := a^1(tu)$. For each $j \in [\alpha_1 d_1]$, $\bar{a}^1_j(\cdot)$ has at most $p$ pieces. On the other hand, $j' \in [d_1] \setminus [[\alpha_1 d_1]]$, $\bar{a}^1_{j'}(\cdot)$ is a linear function and has 1 piece. The input to layer 2 is the weighted sum of $\bar{a}^1_j(\cdot)$'s and $\bar{a}^1_{j'}(\cdot)$'s. Each $\bar{z}^2_k(t) = z^2_k(tu)$ has $\alpha_1 d_1(p-1)+1$ pieces. Now $\alpha_2 d_2$ neurons after the activation $\sigma$ have at most $\alpha_1 d_1 p(p-1)+p$ pieces and $(1-\alpha_2)d_2$ neurons have at most $\alpha_1 d_1(p-1)+1$ pieces. Therefore, the maximum number of pieces from the weighted sum of the second hidden layer neurons is calculated as:

$$\begin{aligned}&\alpha_1 d_1 p(p-1)+p+\sum_{i=1}^{\alpha_2 d_2 \;\; 1}(\alpha_1 d_1 p(p-1)+p-1)+\sum_{i=1}^{(1\;\; \alpha_2)d_2}(\alpha_1 d_1(p-1)+1-1)\\ &=\alpha_1\alpha_2 d_1 d_2 p(p-1)+\alpha_2 d_2(p-1)+\alpha_1(1-\alpha_2)d_1 d_2(p-1)+1\end{aligned}$$

This calculation tells that a 3-layer network has at most $\alpha_1\alpha_2 d_1 d_2 p(p-1)+\alpha_2 d_2(p-1)+\alpha_1(1-\alpha_2)d_1 d_2(p-1)+1$ pieces. If this number is strictly smaller than $N-1$, the network cannot never perfectly fit the given dataset. □

## A.2. Proof of Corollary 2

*Proof.* The proof is a direct consequence of Theorem 1 by replacing $d_1$ and $d_2$ with $\alpha_1 d_1$ and $\alpha_2 d_2$. □

## A.3. Proof of Theorem 2

*Proof.* By setting $p = 2$, we consider the following optimization problem:

$$\begin{aligned}&\max_{\alpha_1,\alpha_2}\ \alpha_1\alpha_2 d_1 d_2+\alpha_1 d_1 d_2+\alpha_2 d_2+1\,,\\ &\text{s.t. } \alpha_1 d_1+\alpha_2 d_2 = B\,.\end{aligned}$$

Constructing the Lagrangian,

$$L(\alpha_1,\alpha_2,\lambda)=\alpha_1\alpha_2 d_1 d_2+\alpha_1 d_1 d_2+\alpha_2 d_2+1-\lambda(\alpha_1 d_1+\alpha_2 d_2-B).$$

By checking the first order condition of the Lagrangian, we get

$$\alpha_1=\frac{B+d_2-1}{2d_1},\qquad \alpha_2=\frac{B-d_2+1}{2d_2}.$$

We examine the determinant of the Hessian $H$ which is defined as

$$H=\begin{bmatrix}0 & d_1 & d_2\\ d_1 & 0 & d_1 d_2\\ d_2 & d_1 d_2 & 0\end{bmatrix}.$$

One can verify that the determinant $\det H > 0$ since $d_1, d_2 > 0$ and $\det H$ has the sign $(-1)^2 = (-1)^{1+1}$ so the critical point $(\frac{B+d_2\;\;1}{2d_1}, \frac{B\;\;d_2+1}{2d_2})$ from the first order condition is indeed a maximum. □

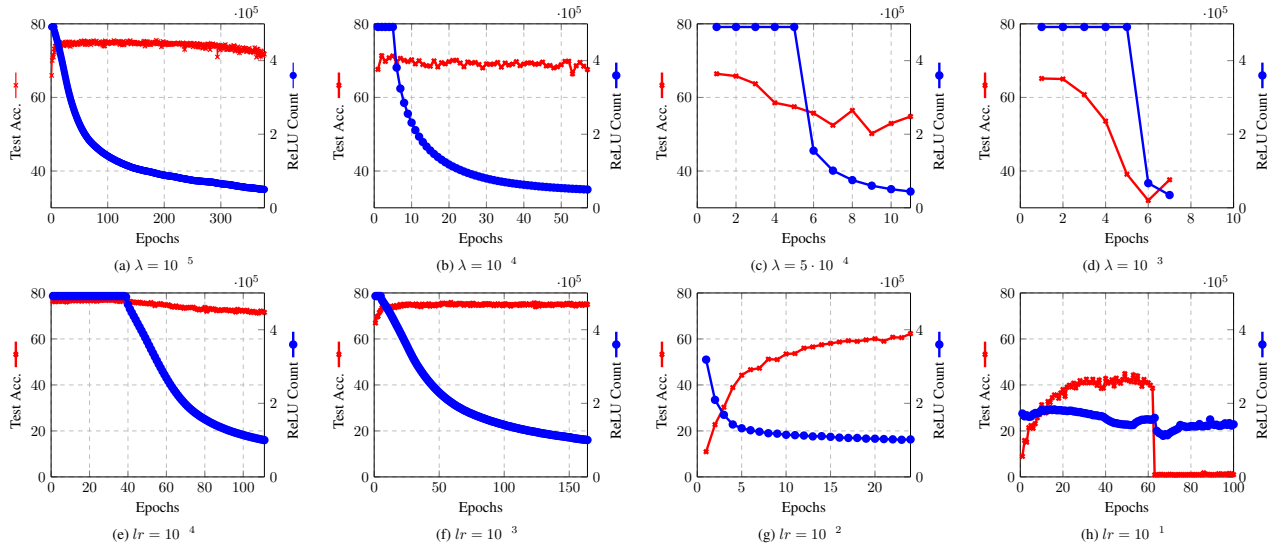



*Figure 5.* Ablation studies on Lasso coefficient $\lambda$ and learning rate $lr$. We test on ResNet18 architecture with CIFAR-100. (a)-(d) show the differences in behavior on test accuracy and ReLU counts with various Lasso coefficient values given the learning rate $lr$ = 0.001 and ADAM optimizer. High initial Lasso coefficients (as shown in (c) and (d)) induces the sparsity on $C$ rapidly and consequently the architecture's performance rapidly degrades as well. Small initial lasso coefficients from (a) and (b) gradually induces the sparsity on $C$ allowing the $W$ to adapt as well in order to maintain the networks' performance. (e)-(f) tracks the test accuracy and ReLU count given the Lasso coefficient $10^{-5}$. We observe the similar trend in learning rate compared to Lasso coefficients; using a small learning rate ((e), (f)) helps to gradually reduce ReLU counts and maintain the test accuracy.

## B. Additional Results

### B.1. Ablation Studies on Hyperparameters

We study the ablation studies on the primary hyperparameters: sparsity factor $\alpha$ and learning rate $lr$. We choose the network ResNet18 with the CIFAR-100 dataset. We first examine the role of the sparsity factor $\alpha$. The targeted ReLU budget here is $B = 50000$, and the plots show the change of ReLU counts and test accuracy along with the training (corresponds to Algorithm 1 line 4-8). We use an ADAM optimizer with a learning rate of $0.001$.

Along with the simultaneous training on the model parameter $W$ and auxiliary parameter $C$, the ideal situation will gradually reduce ReLU counts while maintaining the test accuracy close to the baseline pretrained model. We observe that increasing $\alpha$ induces the sparsity on $C$ faster; however, the test accuracy rapidly drops. On the other hand, the small $\alpha$ (e.g., first and second plot) gradually reduces the ReLU count while almost preserving the test accuracy. We heuristically observed incrementing the Lasso coefficient (line 5-6 in Algorithm 1) allows reaching the desired sparsity level on $C$ with Lasso coefficients initialized with small values.

We also examine the role of learning rate. We use the equivalent hyperparameters as above but with different targeted ReLU budget $B = 100000$ and initial sparsity factor $\alpha = 0.00001$. We observed that high learning rate such as $0.1$ or $0.01$ loses the performance in the beginning of the training and struggles to recover the performance compare to the original baseline performance (around 76%). Our empirical results show that SNL algorithm works well with lower learning rate such as $0.001$ and $0.0001$. SNL algorithm with $lr = 0.001$ terminates the simultaneous training with the test accuracy (before the finetuning) $75.04\%$ while $lr = 0.0001$ achieves the $71.55\%$.

### B.2. Ablation Studies on Algorithmic Components

We study the variant of SNL algorithm by changing the algorithmic components. First, we consider zeroing out non-activated output instead of passing through the pre-activated inputs. In this case, the output, $\sigma_{c^i}(\cdot)$ at the i$^{\text{th}}$ layer, can be expressed as:

$$a^i_{c^i} = c^i \odot \sigma(z^i) + (\mathbf{1} - c^i) \odot \mathbf{0} \tag{7}$$

Equation 7 is equivalent to feature map sparsification where feature maps are zeroed out pixel-wise rather than filter-wise. Second, we perform the SNL algorithm from randomly initialized network (remark: SNL starts from the pretrained network).

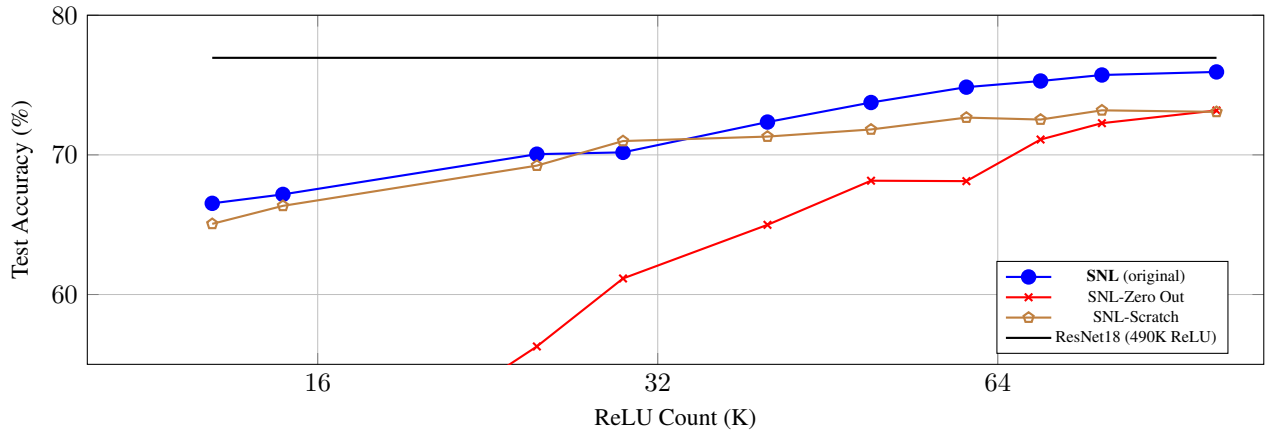



*Figure 6.* **Comparison between SNL and its variants on CIFAR-100**. All experiments in this plot uses ResNet18.

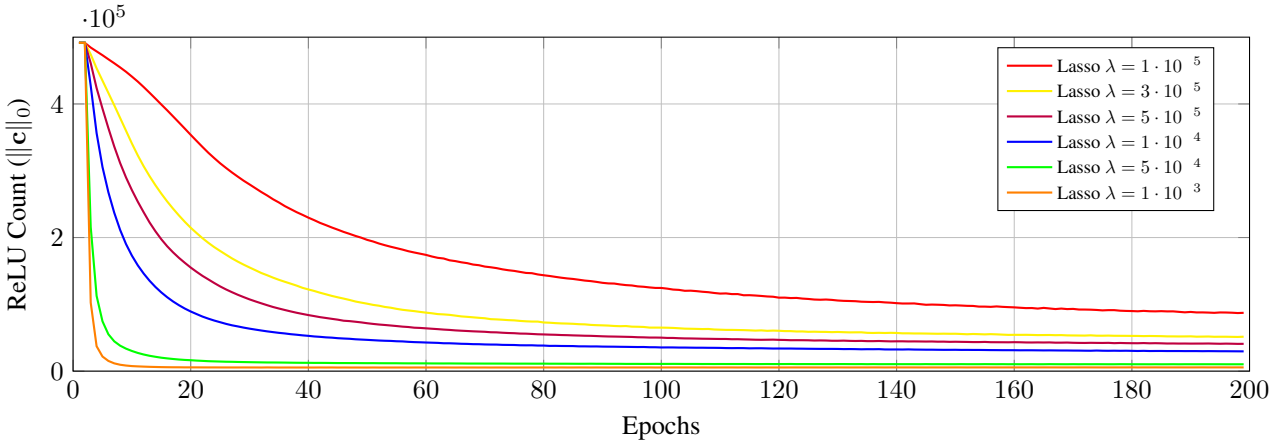



*Figure 7.* **Effect of the Lasso Coefficient on supports of c.** We track the number of supports in the auxiliary parameter $\mathbf{c}$ for various Lasso coefficients. See that the magnitude of Lasso coefficient ($\lambda$) determines the convergence rate and the support of $\mathbf{c}$.

In Figure 6, "SNL-Zero-Out" and "SNL-Scratch" correspond to feature map sparsification and the SNL network trained from scratch, respectively. For this comparison, we train a ResNet18 architecture with the CIFAR100 dataset. A black horizontal line is the reference performance (76.95% test accuracy) of the original ResNet18 network with $490K$ ReLU. We observe that the "SNL-Zero-Out" performance degrades heavily in the sparse ReLU regime ($< 50K$). Interestingly, "SNL-Scratch" showed a competitive performance to SNL while the original SNL consistently performed better in all cases except for one data point. Overall, this plot gives further evidence that the current SNL algorithm finds better performing architectures for a given ReLU budget over the SNL variants suggested by the reviewers. Finally, note that we incorporate knowledge distillation (KD) during the training, as clearly stated in the main paper (Line 268). We will add these results and discussion in the final draft.

### B.3. Role of the Lasso Coefficient

From our empirical observations, the magnitude of the Lasso coefficient ($\lambda$) determines the final (anti) sparsity of the slope parameters, and the convergence rate. We conduct experiments with a pre-trained ResNet18 architecture trained on CIFAR-100. We modify the SNL algorithm by removing the "if-else" condition, gradually increasing the Lasso coefficient, and track ReLU counts given by the auxiliary parameter $\mathbf{c}$ until 200 epochs. The other hyperparameters remain the same as our reported experiments: ADAM, $lr = 0.001$, weight decay of 0.0005. Figure 7 shows that the magnitude of the Lasso coefficient directly contributes to the size of the support of $\mathbf{c}$ and the convergence rate. These observations motivate Line 6-7 in SNL, which uses homotopy-style optimization to gradually increases the Lasso coefficient to reach the desired sparsity while ensuring mild accuracy drop (also see Figure 5 in the main paper showing that large $\lambda$ hurts test accuracy).

*Table 3.* CIFAR-10 Comparison.

| | Methods | #ReLUs (K) | Test Acc. (%) | Acc./ReLU |
|---|---|---|---|---|
| 100K ReLU | SNL[#] | 12.9 | 88.23 | 6.840 |
| | SNL[#] | 25.0 | 90.88 | 3.635 |
| | SNL[#] | 60.0 | 92.63 | 1.544 |
| | DeepReDuce | 36.0 | 88.5 | 2.458 |
| | DeepReDuce | 80.0 | 90.5 | 1.131 |
| | CryptoNAS | 86.0 | 91.28 | 1.061 |
| 400K | SNL[*] | 240.0 | 94.24 | 0.4712 |
| | SNL[*] | 300.0 | 95.06 | 0.317 |
| | CryptoNAS | 344.0 | 94.04 | 0.273 |

[#] Starts with pretrained ResNet18.
[*] Starts with pretrained Wide-ResNet 22-8.

### B.4. Additional PI Results

Table 3 and Table 4 shows the PI comparison on CIFAR-10 and Tiny-ImageNet, respectively. Table 5 includes the additional PI results with pretrained ResNet34 network. Table 6 contains the SNL results with channel-wise method demonstrated in Section 4.

*Table 4.* Tiny-ImageNet Comparison

| | Methods | #ReLUs (K) | Test Acc. (%) | Online Lat. (s) | Acc./ReLU |
|---|---|---|---|---|---|
| 100K ReLU | SNL[#] | 59.1 | 54.24 | 1.265 | 0.918 |
| | SNL[#] | 99.6 | 58.94 | 2.117 | 0.592 |
| | DeepReDuce | 57.35 | 53.75 | 1.85 | 0.937 |
| | DeepReDuce | 98.3 | 55.67 | 2.64 | 0.566 |
| | Sphynx | 102.4 | 48.44 | 2.350 | 0.473 |
| 300K ReLU | SNL[#] | 198.1 | 63.39 | 4.183 | 0.320 |
| | SNL[#] | 298.2 | 64.04 | 6.281 | 0.215 |
| | DeepReDuce | 196.6 | 57.51 | 4.61 | 0.293 |
| | DeepReDuce | 393.2 | 61.65 | 7.77 | 0.157 |
| | Sphynx | 204.8 | 53.51 | 4.401 | 0.261 |
| 1000K ReLU | SNL[*] | 488.8 | 64.42 | 10.281 | 0.132 |
| | DeepReDuce | 917.5 | 64.66 | 17.16 | 0.070 |
| | Sphynx | 614.4 | 60.76 | 12.548 | 0.099 |

[#] Starts with pretrained ResNet18.
[*] Starts with pretrained Wide-ResNet 22-8.

*Table 5.* SNL on ResNet34 Networks on CIFAR-100, Tiny-ImageNet

| | Methods | #ReLUs (K) | Test Acc. (%) | Online Lat. (s) | Acc./ReLU |
|---|---|---|---|---|---|
| CIFAR-100 | SNL | 14.9 | 67.08 | 0.339 | 4.502 |
| | SNL | 25.0 | 69.68 | 0.551 | 2.787 |
| | SNL | 30.0 | 70.99 | 0.656 | 2.366 |
| | SNL | 50.0 | 72.91 | 1.075 | 1.458 |
| | SNL | 80.0 | 74.19 | 1.705 | 0.927 |
| | SNL | 99.9 | 74.76 | 2.122 | 0.748 |
| | SNL | 118.0 | 75.32 | 2.502 | 0.638 |
| | SNL | 197.1 | 76.03 | 4.161 | 0.385 |
| Tiny-ImageNet | SNL | 200.0 | 62.49 | 4.231 | 0.312 |
| | SNL | 300.0 | 63.99 | 6.329 | 0.213 |
| | SNL | 400.0 | 65.31 | 8.426 | 0.163 |
| | SNL | 500.0 | 65.34 | 10.524 | 0.131 |

*Table 6.* CIFAR-100 Results on Channel-wise SNL.

| Methods | #ReLUs (K) | Test Acc. (%) | Online Lat. (s) | Acc./ReLU |
|---|---|---|---|---|
| Channel-wise SNL[#] | 29.0 | 68.26 | 0.628 | 2.354 |
| Channel-wise SNL[#] | 39.3 | 70.52 | 0.843 | 1.794 |
| Channel-wise SNL[#] | 49.8 | 71.56 | 1.065 | 1.437 |
| Channel-wise SNL[*] | 77.7 | 73.47 | 1.650 | 0.940 |
| Channel-wise SNL[*] | 117.3 | 74.33 | 2.481 | 0.634 |
| Channel-wise SNL[*] | 200.0 | 77.45 | 4.216 | 0.387 |

[#] Starts with pretrained ResNet18.
[*] Starts with pretrained Wide-ResNet 22-8.