Scaling Up Exact Neural Network Compression by ReLU Stability

Thiago Serra
Bucknell University
Lewisburg, PA, United States
thiago.serra@bucknell.edu

Abhinav Kumar
Michigan State University
East Lansing, MI, United States
kumarab6@msu.edu

Xin Yu
The University of Utah
Salt Lake City, UT, United States
xin.yu@utah.edu

Srikumar Ramalingam
Google Research
New York, NY, United States
rsrikumar@google.com

Abstract

We can compress a neural network while exactly preserving its underlying functionality with respect to a given input domain if some of its neurons are stable. However, current approaches to determine the stability of neurons with Rectified Linear Unit (ReLU) activations require solving or finding a good approximation to multiple discrete optimization problems. In this work, we introduce an algorithm based on solving a single optimization problem to identify all stable neurons. Our approach is on median 100 times faster than the state-of-art method, which allows us to explore exact compression on deeper (5×100) and wider (2×800) networks within minutes. For classifiers trained under an amount of ℓ_1 regularization that does not worsen accuracy, we can remove up to 40% of the connections.

1 Introduction

For the past decade, the computing requirements associated with state-of-art machine learning models have grown faster than typical hardware improvements [5]. Although those requirements are often associated with training neural networks, they also translate into larger models, which are challenging to deploy in modest computational environments, such as in mobile devices.

Meanwhile, we have learned that the expressiveness of the models associated with neural networks—when measured in terms of their number of linear regions —grows polynomially on the number of neurons and occasionally exponentially on the network depth [64; 61; 68; 76; 35; 36]. Hence, we may wonder if the pressing need for larger models could not be countered by such gains in model

Preprint. Under review.

complexity. Namely, if we could not represent the same model using a smaller neural network. More specifically, we consider the following definition of equivalence [74]:

Definition 1. *Two neural networks $\mathcal{N}_1$ and $\mathcal{N}_2$ with associated functions $\boldsymbol{f}_1 : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^m$ and $\boldsymbol{f}_2 : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^m$ are local equivalent with respect to a domain $\mathbb{D} \subseteq \mathbb{R}^{n_0}$ if $\boldsymbol{f}_1(x) = \boldsymbol{f}_2(x)\ \forall x \in \mathbb{D}$.*

There is an extensive literature on methods for compressing neural networks [18; 11], which is aimed at obtaining smaller networks that are nearly as good as the original ones. These methods generally produce networks that not equivalent, and thus require retraining the neural network for better accuracy. They may also disproportionately affect some inputs more than others [42].

Compressing a neural network while preserving its associated function is a relatively less explored topic, which has been commonly referred to as *lossless compression* [74; 77]. However, that term has also been used for the more general case in which the overall accuracy of the compressed network is no worse than that of the original network regardless of equivalence [89]. Hence, we regard *exact compression* as a more appropriate term when equivalence is preserved.

Exact compression has distinct benefits and challenges. On the one hand, there is no need for retraining and no risk of disproportionately affecting some inputs more than others. On the other hand, optimization problems that are formulated for exact compression need to account for any valid input as opposed to relying on a sample of inputs. In this paper, we focus on how to scale such an approach to a point in which exact compression starts to become practical for certain applications.

In particular, we introduce and evaluate a faster algorithm for exact compression based on identifying all neurons with Rectified Linear Unit (ReLU) activation that have a linear behavior, which are denoted as *stable*. In other words, those are the neurons for which the mapping of inputs to outputs is always characterized by a linear function, which is either the constant value 0 or the preactivation output. We can remove or merge such neurons—and even entire layers in some cases—while obtaining a smaller but equivalent neural network. Our main contributions are the following:

(i) We propose an algorithm based on solving a single Mixed-Integer Linear Programming (MILP) formulation to verify the stability of all neurons of a feedforward neural network with ReLU activations, which is faster than solving MILP formulations for every neuron—either optimally [84] or approximately [74]. Compared to [74], the median improvement is of 100 times—and in fact greater in larger networks.

(ii) We reduce the runtime with a GPU-based preprocessing step that identifies neurons that are not stable with respect to the training set. The median improvement for that part alone is of 3.2 times.

(iii) We outline an algorithm that leverages (i) to perform all compressions once per layer instead of once per stable neuron [74], and then prove the correctness of its most elaborate steps.

(iv) We leverage the scalability of our approach to investigate exact compressibility on classifiers that are deeper (5×100) and wider (2×800) than previously studied in [74] (2×100). We show that approximately 20% of the neurons and 40% of the connections can be removed from MNIST classifiers trained with an amount of ℓ_1 regularization that does not worsen accuracy.

2 Related work

There are many pruning methods for sparsifying or reducing the size of neural networks by removing connections, neurons, or even layers. They are justified by the significant redundancy among parameters [22] and the better generalization bounds of compressed networks [8; 92; 81; 82].

Surveys such as [11] note that these methods are typically based on a tradeoff between model efficiency and quality: the models of compressed neural networks tend to have a comparatively lower accuracy, save some exceptions [34; 89; 81]. Nevertheless, such compression leads to networks in which the loss in accuracy is disproportionately distributed across classes and more severe in a fraction of them; the most impacted inputs are those that the original network could not classify well; and the overall robustness to noise or adversarial examples is diminished [42].

To make up for model changes and potential accuracy loss, one may rely on a three-step procedure consisting of (1) training the neural network; (2) compression; and (3) retraining. Nevertheless, the scope of compression methods is seldom restricted to the second step. For example, the compressibility of a neural network hinges on how it was trained, with regularizations such as ℓ_1 and ℓ_2 often used to make part of the network parameters negligible in magnitude—and hopefully in impact as well.

In fact, the two main—and recurring—themes in this topic are pruning connections when the corresponding parameters are sufficiently small [37; 62; 44; 34; 33; 54; 28; 30; 83] and when the impact of the connection on the loss function is sufficiently small [51; 38; 49; 60; 24; 90; 91; 53; 85; 86]. The main issue with the first approach is that small weights may nevertheless be important, although it is possible to empirically quantify their impact on the loss function [71]. The main issue with the second approach is the computational cost of calculating the second-order derivatives of the loss function in deep networks, which has lead to many approaches for approximating such values.

Overlapping with such approximations, there is a growing literature on casting neural network compression as an optimization problem [40; 58; 1; 90; 74; 25]. Most often, these formulations aim to minimize the impact of the compression on how the neural network performs on the training set.

Other lines of work and overlapping themes in neural network compression include combining similar neurons [78; 59; 80; 81]; low-rank approximation, factorization, and random projection of the weight matrices [43; 23; 48; 8; 87; 79; 85; 81; 82; 55]; and statistical tests on the relevance of a connection to network output [89]. Many recent approaches focus on pruning at initialization instead of after training [53; 52; 86; 83; 29] as well as on what parameters to use when these networks are retrained [28; 57; 69].

Exact compression was only recently explored for fully-connected feedforward neural networks [74] and graph neural networks [77]. Nevertheless, we may associate it with the literature on neural network equivalency, which includes verifying that networks are equivalent [63; 15], identifying operations that produce equivalent networks [41; 16; 46; 47; 67], reconstructing networks from their outputs [3; 4; 26; 2; 70], and evaluating the effect of redundant representations on training [10; 66].

3 Setting and notation

We consider fully-connected feedforward neural networks with L hidden layers, in which we denote n_l as the number of units—or width—of layer $l \in \mathbb{L} := \{1, 2, \ldots, L\}$ and x_i^l as the output of the i-th unit of layer l for $i \in \{1, 2, \ldots, n_l\}$. For uniformity, we denote $\boldsymbol{x}^0 \in \mathbb{R}^{n_0}$ as the network input. We denote the output of the i-th unit of layer l as $x_i^l = \sigma^l(y_i^l)$, where the pre-activation output $y_i^l := \boldsymbol{w}_i^l \cdot x^{l-1} + b_i^l$ is defined by the learned weights $\boldsymbol{w}_i^l \in \mathbb{R}^{n_{l-1}}$ and the bias $b_i^l \in \mathbb{R}$ of the unit as well as the activation function $\sigma^l : \mathbb{R} \to \mathbb{R}$ associated with layer l, which is $\sigma^l(u) = \max\{0, u\}$—the ReLU [32]. The output layer may have a different structure, such as the softmax layer [13], which is nevertheless irrelevant for our purpose of compressing the hidden layers. For every layer $l \in \mathbb{L}$, let $\boldsymbol{W}^l = [\boldsymbol{w}_1^l \boldsymbol{w}_2^l \ldots \boldsymbol{w}_{n_l}^l]^T$ be the matrix of weights, $\boldsymbol{W}_{\mathbb{S}}^l$ be a submatrix of $\boldsymbol{W}^l$ consisting of the rows in set $\mathbb{S}$, and $\boldsymbol{b}^l = [b_1^l b_2^l \ldots b_{n_l}^l]^T$ be the vector of biases. Finally, let $\boldsymbol{I}_m(\mathbb{S})$ be an $m \times m$ diagonal matrix in which the i-th diagonal element is 1 if $i \in \mathbb{S}$ and 0 if $i \notin \mathbb{S}$.

4 Identifying stability for exact compression

This section explains the concept of stability and describes how MILP has been used to identify stable neurons. If the output of neuron i in layer l is always linear on its inputs, we say that the neuron is stable. This happens in two ways for the ReLU activation. When $x_i^l = 0$ for any valid input, which implies that $y_i^l \leq 0$, we say that the neuron is *stably inactive*. When $x_i^l = y_i^l$ for any valid input, which implies that $y_i^l \geq 0$, we say that the neuron is *stably active*.

The qualifier *valid* is essential since not every input may occur in practice. If $\boldsymbol{w}_i^l \neq \boldsymbol{0}$, there are nonempty halfspaces on $\boldsymbol{x}^{l-1}$ that would make that neuron active or inactive, $\{\boldsymbol{x}^{l-1} : \boldsymbol{w}_i^l \cdot \boldsymbol{x}^{l-1} + b_i^l \leq 0\}$ and $\{\boldsymbol{x}^{l-1} : \boldsymbol{w}_i^l \cdot \boldsymbol{x}^{l-1} + b_i^l \geq 0\}$, but it is possible that valid inputs only map to one of them. For the first layer, we only need to account for the valid inputs to the neural network. For example, the domain of a network trained on the MNIST dataset is $\{\boldsymbol{x}^0 : \boldsymbol{x}^0 \in [0, 1]^{784}\}$ [50]. For the remaining hidden layers, we also account for the combinations of outputs that can be produced by the preceding layers given their valid inputs and parameters. Hence, assessing stability is no longer straightforward.

We can determine if a neuron of a trained neural network is stable by solving optimization problems to maximize and minimize its preactivation output [84]. The main decision variables in these problems are the inputs for which the preactivation output is optimized. Consequently, there is also a decision variable associated with the output of every neuron, in addition to other variables described below.

MILP formulation of a single neuron For every neuron i of layer l, we map every input vector $\boldsymbol{x}^{l-1}$ to the corresponding output x_i^l through a set of linear constraints that also include a binary variable z_i^l denoting if the unit is active or not, a variable for the pre-activation output y_i^l, a variable $\chi_i^l := \max\{0, -y_i^l\}$ denoting the output of a complementary fictitious unit, and positive constants M_i^l and μ_i^l that are as large as x_i^l and χ_i^l can be. The constraints below are explained in Appendix A.

$$\boldsymbol{w}_i^l \cdot \boldsymbol{x}^{l-1} + b_i^l = y_i^l = x_i^l - \chi_i^l \tag{1}$$

$$0 \leq x_i^l \leq M_i^l z_i^l \tag{2}$$

$$0 \leq \chi_i^l \leq \mu_i^l (1 - z_i^l) \tag{3}$$

$$z_i^l \in \{0, 1\} \tag{4}$$

Using MILP to determine stability Let $\mathbb{X} \subset \mathbb{R}^{n_0}$ be the set of valid inputs for the neural network, which we may assume to be bounded in every direction. We can obtain the interval $[\underline{\mathcal{Y}}_i^{l'}, \overline{\mathcal{Y}}_i^{l'}]$ for the preactivation output $y_i^{l'}$ of neuron i in layer l' by solving the following optimization problems [84]:

$$\underline{\mathcal{Y}}_i^{l'} := \left\{ \min \boldsymbol{w}_i^{l'} \cdot \boldsymbol{x}^{l'-1} + b_i^{l'} : \boldsymbol{x}^0 \in \mathbb{X}; (1) - (4) \ \forall l \in [l'-1], i \in [n_l] \right\} \tag{5}$$

$$\overline{\mathcal{Y}}_i^{l'} := \left\{ \max \boldsymbol{w}_i^{l'} \cdot \boldsymbol{x}^{l'-1} + b_i^{l'} : \boldsymbol{x}^0 \in \mathbb{X}; (1) - (4) \ \forall l \in [l'-1], i \in [n_l] \right\} \tag{6}$$

When $\overline{\mathcal{Y}}_i^{l'} \leq 0$, then $x_i^{l'} = 0$ for every $x^0 \in \mathbb{X}$ and the neuron is stably inactive. When $\underline{\mathcal{Y}}_i^{l'} \geq 0$, then $x_i^{l'} = y_i^{l'}$ for every $x^0 \in \mathbb{X}$ and the neuron is stably active.

Variations of the formulations above have been proposed for diverse tasks over neural networks, such as verifying them [17], embedding their model into a broader decision-making problem [73; 9; 21], and measuring their expressiveness [76]. When stable units are identified, other optimization problems over trained neural networks become easier to solve [84]. For example, weight regularization can induce neuron stability and facilitate adversarial robustness verification [88]. There is extensive work on the properties of such formulations and methods to solve them more effectively [27; 7; 12; 75; 6].

For the purpose of identifying stable neurons, however, it is not scalable to analyze large neural networks by solving such optimization problems for every neuron [84]—or even by just approximately solving each of them to ensure that $\overline{\mathcal{Y}}_i^{l'} \leq 0$ or $\underline{\mathcal{Y}}_i^{l'} \geq 0$ [74].

5 A new algorithm for exact compression

Based on observations discussed in what follows (I to III), we propose a new MILP formulation to identify stable neurons (Section 5.1), means to generate feasible solutions while the formulation is solve (Section 5.2), a preprocessing step to reduce the effort to solve the formulation (Section 5.3), and a compression algorithm exploiting all stable neurons in each layer at once (Section 5.4).

5.1 A new MILP formulation

Consider the two observations below and their implications:

I: The overlap between optimization problems Although previous approaches require solving many optimization problems, their formulations are all very similar: we maximize or minimize the same objective function for each neuron, the feasible set of the problems for each layer are the same, and they are contained in the feasible set of problems for the subsequent layers.

II: Proving stability is harder than disproving it Certifying that a neuron is stable is considerably more complex than certifying that a neuron is *not* stable. For the former, we need to exhaustively show that all inputs lead to the neuron always being active or always being inactive, which can be achieved by solving (5) and (6) for every neuron. For the latter, we just need a pair of inputs to the neural network such that the neuron is active with one of them and inactive with the other.

Therefore, we consider the problem of finding an input that serves as a certificate of a neuron not being stable to as many neurons of unknown classification as possible. For that purpose, we define a decision variable $p_i^l \in \{0, 1\}$ to denote if an input activates neuron i in layer l. Likewise, we define a decision variable $q_i^l \in \{0, 1\}$ to denote if an input does not activate neuron i in layer l. Furthermore, we restrict the scope of the problem to states that have not been previously observed by using $\mathbb{P}^l \subseteq \{1, \ldots, n_l\}$ as the set of neurons in layer l for which there is no known input that activates the neuron. Likewise, we use $\mathbb{Q}^l \subseteq \{1, \ldots, n_l\}$ as the set of neurons in layer l for which there is no known input that does not activate the neuron. For brevity, let $\boldsymbol{P} := (\mathbb{P}^1, \ldots, \mathbb{P}^L)$ and $\boldsymbol{Q} := (\mathbb{Q}^1, \ldots, \mathbb{Q}^L)$ characterize an instance of such optimization problem, which is formulated as follows:

$$\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = \max \quad \sum_{l \in \mathbb{L}} \left(\sum_{i \in \mathbb{P}^l} p_i^l + \sum_{i \in \mathbb{Q}^l} q_i^l \right) \tag{7}$$

$$\text{s.t.} \quad \boldsymbol{x}^0 \in \mathbb{X} \tag{8}$$

$$(1) - (4) \ \forall l \in \mathbb{L}, i \in [n_l] \tag{9}$$

$$0 \leq p_i^l \leq z_i^l \ \forall l \in \mathbb{L}, i \in \mathbb{P}^l \tag{10}$$

$$0 \leq q_i^l \leq 1 - z_i^l \ \forall l \in \mathbb{L}, i \in \mathbb{Q}^l \tag{11}$$

$$p_i^l, q_i^l \in \{0, 1\} \tag{12}$$

Note that constraint (12) is actually not necessary. We refer to Appendix B for more details.

The formulation above yields an input that maximizes the number of neurons with an activation state that has not been previously observed. The following results show that it entails an approach in which no more than $N + 1$ such formulations are solved. We refer to Appendix C for the proofs.

Proposition 1. *If $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = 0$, then every neuron $i \in \mathbb{P}^l$ is stably inactive and every neuron $i \in \mathbb{Q}^l$ is stably active.*

Corollary 2. *The stability of all neurons of a neural network can be determined by solving formulation* (7)–(12) *at most $N + 1$ times, where $N := \sum_{l \in \mathbb{L}} n_l$.*

Those results imply that we can iteratively solve the new formulation as part of an algorithm to identify all stable neurons. In fact, we can determine the stability of the entire neural network with a single call to the MILP solver. Except for the last time that formulation (7)–(12) is solved, there is no need to solve it to optimality: any solution with a positive objective function value can be used to reduce the number of unobserved states. Hence, all that we need is a way to inspect every feasible solution obtained by the MILP solver and then remove the solutions in which either $p_i^l = 1$ or $q_i^l = 1$ for states that were already observed. Both of those needs can be addressed in fully-fledged MILP solvers by implementing a lazy constraint callback. We refer to Appendix D for more details. When we finally reach $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = 0$, the correctness of the MILP solver serves as a certificate of the stability of those remaining neurons. The resulting algorithm is described in Appendix E.

5.2 Inducing feasible MILP solutions

The runtime with a single solver call depends on the frequency with which feasible solutions are obtained. Although at most $N + 1$ optimal solutions would suffice if we were to make consecutive calls to the solver until $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = 0$, we should not expect the same from the first $N + 1$ feasible solutions found by the MILP solver while using the lazy constraint callback because they may not have a positive objective function value due to the p_i^l and q_i^l variables that have been fixed to 0. On top of that, obtaining a feasible solution for an MILP formulation is NP-complete [19].

III: Finding feasible solutions to MILP formulations of neural networks is easy To any valid input of the neural network there is a corresponding solution of the MILP formulation: the neural network input implies which neurons are active and what is their output when active [27].

Although any random input would suffice, we have found that it is better in practice to use inputs indirectly generated by the MILP solver. Namely, we can use the solution of the Linear Programming (LP) relaxation, which is solved at least once per branch-and-bound node. The LP relaxation is obtained from the MILP formulation by relaxing its integrality constraints. In the case of binary variables with domain $\{0, 1\}$, that consists of relaxing the domain of such variables to the continuous interval $[0, 1]$. We use the values of $\boldsymbol{x}^0$ in the solution of the LP relaxation as the network input, and thus obtain a feasible MILP solution by replacing the values of the other variables—which may be fractional for the decision variables with binary domains—by the values implied by fixing $\boldsymbol{x}^0$. However imprecise due to the relaxation of the binary domains, the input defined by the optimal solution of the LP relaxation may intuitively guide us toward maximizing the objective function.

5.3 Preprocessing

In addition to generating feasible MILP solutions at every node of the branch-and-bound tree during the solving process, we also evaluate the training set on the trained neural network to reduce the number of states that need to be search for by the MILP solver. By using GPUs, this step can be completed in few seconds for all the experiments performed.

5.4 Compressing the neural network

Algorithm 1 Performs exact compression of a neural network

Input: neural network $\left(L, \left\{(n_l, \boldsymbol{W}^l, \boldsymbol{b}^l)\right\}_{l \in \mathbb{L}}\right)$ and stable neurons $\left(\left\{(\mathbb{P}^l, \mathbb{Q}^l)\right\}_{l \in \mathbb{L}}\right)$
for $l \leftarrow 1$ **to** L **do**
    if $|\mathbb{P}^l| = n_l$ **then**
        find output $\overline{\boldsymbol{x}}^L$ for an arbitrary input $\overline{\boldsymbol{x}}^0 \in \mathbb{X}$
        remove all layers except L, which becomes 1
        $\boldsymbol{W}^1 \leftarrow \boldsymbol{0}$ and $\boldsymbol{b}^L \leftarrow \overline{\boldsymbol{x}}^L$
        break
    else if $|\mathbb{P}^l| + |\mathbb{Q}^l| = n_l$ and $l < L$ **then**
        $\boldsymbol{W}^{l+1} \leftarrow \boldsymbol{W}^{l+1} \boldsymbol{I}_{n_l}(\mathbb{Q}^l) \boldsymbol{W}^l$ and $\boldsymbol{b}^{l+1} \leftarrow \boldsymbol{W}^{l+1} \boldsymbol{I}_{n_l}(\mathbb{Q}^l) \boldsymbol{b}^l + \boldsymbol{b}^{l+1}$
        remove layer l
    else if $l < L$ **then**
        $r \leftarrow \text{rank}\left(\boldsymbol{W}^l_{\mathbb{Q}^l}\right)$
        if $r < |\mathbb{Q}^l|$ and $l < L$ **then**
            find $\overline{\mathbb{Q}} \subset \mathbb{Q}^l$ such that $r = |\overline{\mathbb{Q}}| = \text{rank}\left(\boldsymbol{W}^l_{\overline{\mathbb{Q}}}\right)$
            for every $i \in \mathbb{Q}^l \setminus \overline{\mathbb{Q}}$ **do**
                find $\{\alpha^i_j\}_{j \in \overline{\mathbb{Q}}}$ such that $\boldsymbol{w}^l_i = \sum_{j \in \overline{\mathbb{Q}}} \alpha^i_j \boldsymbol{w}^l_j$
            end for‘
            for $k \leftarrow 1$ **to** n_{l+1} **do**
                for every $j \in \overline{\mathbb{Q}}$ **do**
                    $w^{l+1}_{kj} \leftarrow w^{l+1}_{kj} + \sum_{i \in \mathbb{Q}^l \setminus \overline{\mathbb{Q}}} \alpha^i_j w^{l+1}_{ki}$
                end for
                $b^{l+1}_k \leftarrow b^{l+1}_k + \sum_{i \in \mathbb{Q}^l \setminus \overline{\mathbb{Q}}} w^{l+1}_{ki} \left(b^l_i - \sum_{j \in \overline{\mathbb{Q}}} \alpha^i_j b^l_j\right)$
            end for
            remove from layer l every neuron $i \in \mathbb{Q}^l \setminus \overline{\mathbb{Q}}$
            remove from layer l every neuron $i \in \mathbb{P}^l$
        end if
    end if
end for

Algorithm 1 leverages neuron stability for exactly compressing neural networks. We describe next each form of compression contained in the algorithm. For ease of explanation, they are in reverse order of appearance. These compression operations are the same as in [74], but performed once per layer instead of once per neuron. In comparison to that work, the order of the operations is such that (i) neurons are not removed or merged if the entire layer is going to be folded; and (ii) special cases such as a neuron with weight vector $\boldsymbol{w}^l_i = \boldsymbol{0}$ do not need to be considered apart. For the most elaborate operations, we prove their correctness when applied to the entire layer. We refer to Appendix F for the proofs.

Removal of stably inactive neurons This operation is performed in line 25. Since the output of stably inactive neurons is always 0, we remove those neurons without affecting subsequent computations. The case in which an entire layer is stably inactive is considered separately.

Merging of stably active neurons This operation is performed between lines 12 and 24. We use the following results to show how stably active neurons can be merged.

Proposition 3. *Let $\mathbb{S}$ be a set of stably active neurons in layer l. If $r := rank(\boldsymbol{W}^l_S) < |S|$ and let $\mathbb{T} \subset \mathbb{S}$ be a subset of those neurons for which $rank(\boldsymbol{W}^l_{\mathbb{T}}) = r$, then the output of the neurons in $\mathbb{S} \setminus \mathbb{T}$ is an affine function on the output of the neurons in $\mathbb{T}$.*

Corollary 4. *If $\mathbb{S}$, $\mathbb{T}$, and l are such as in Proposition 3, then the pre-activation output of the neurons in layer $l+1$ is an affine function on the outputs of all neurons from layer l with exception of the neurons in $\mathbb{T}$.*

In Algorithm 1, we use relationships implied by the proof of Corollary 4 with $\mathbb{S} = \mathbb{Q}^l$ and $\mathbb{T} = \overline{\mathbb{Q}}$ to merge stably active neurons. By adjusting the biases of the neurons in the next layer as well as the weights connecting every neuron in $\overline{\mathbb{Q}}$ with the neurons in the next layer, we assign a weight of 0 to the connections between every neuron in $\mathbb{Q}^l \setminus \overline{\mathbb{Q}}$ and the neurons in the next layer. Hence, we simply remove all neurons in $\mathbb{Q}^l \setminus \overline{\mathbb{Q}}$ after adjusting those network parameters.

The case in which an entire layer is stably active—either before any compression is applied or once stably inactive neurons are removed—is also considered separately.

Folding of stable layers This operation is performed between lines 8 and 10. We use the following results to show that stable layers can be folded in a single step.

Proposition 5. *If all the neurons of layer $l \in \mathbb{L} \setminus \{L\}$ are stably active, then the pre-activation output of layer $l+1$ is an affine function on the inputs of layer l.*

Corollary 6. *If all neurons of layer $l \in \mathbb{L} \setminus \{L\}$ are stable, then the pre-activation output of layer $l+1$ is an affine function on the inputs of layer l.*

In Algorithm 1, we use relationships implied by the proof of Corollary 6 with $\mathbb{S} = \mathbb{Q}^l$ to fold stable layers. By adjusting the biases and the weights of layer $l+1$, that layer directly uses the outputs from layer $l-1$.

Although the steps above would apply if a layer is stably inactive, that case deserves separate consideration.

Collapse of a network with stably inactive layers This operation is performed between lines 3 and 7. If layer $l \in \mathbb{L}$ are stably inactive, then $\boldsymbol{x}^l = 0$ for any input $\boldsymbol{x}^0 \in \mathbb{X}$ and thus the value of $\boldsymbol{x}^L$ is constant. Hence, we collapse layers 1 to $L-1$ by making the output of the remaining layer constant and equal to such value of $\boldsymbol{x}^L$.

5.5 On the complexity of the new algorithm

While our algorithm requires solving fewer optimization problems than in [74], the dependence on solving a single NP-hard problem—such as MILP formulations in general—implies an exponential worst-case complexity. Nevertheless, the progress of MILP in the past decades makes it possible to solve considerably large problems with state-of-art MILP solvers. In that context, we show the performance gains of our algorithm empirically.

6 Experimental results

We trained and evaluated the compressibility of classifiers for the datasets MNIST [50], CIFAR-10 [45], and CIFAR-100 [45] with and without ℓ_1 weight regularization, which is known to induce stability [84]. We refer to Appendix G for details on environment and implementation.

We use the notation $L \times n$ for the architecture of L hidden layers with n neurons each. We started at $L = 2$ and $n = 100$, and then doubled the width n or incremented the depth L until the majority of the runs for MNIST classifiers for any configuration timed out after 10,800 seconds. With preliminary

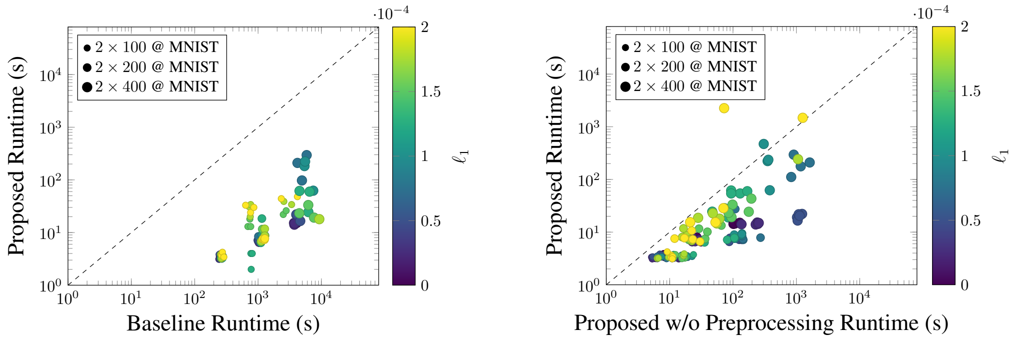


Figure 1: **Comparison of runtimes (in seconds) to identify all stable neurons.** On the left, we compare the proposed approach against the baseline from [74]. On the right, we compare the proposed approach with preprocessing against the proposed approach without preprocessing.

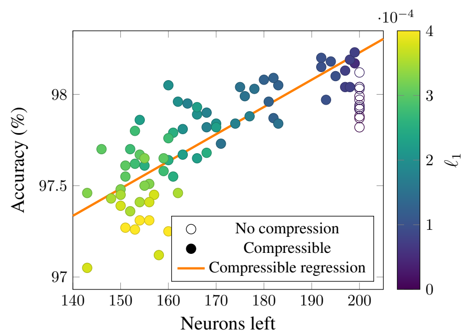


Figure 2: **Relationship between size of compressed neural network and accuracy on** 2×100 **MNIST classifiers.** The coefficient of determination (R^2) for linear regression obtained on the relationship between accuracy and neurons left for compressible networks is 61%.

runs, we made sure that the values chosen for ℓ_1 spanned from those for which accuracy is improving as ℓ_1 increases until those for which the accuracy starts decreasing. We trained and evaluated neural networks with 5 different random initialization seeds for each choice of ℓ_1. The amount of regularization used did not stabilize the entire layer. We refer to Appendix H for additional figures and tables with complete results.

Fig. 3 illustrates the average accuracy and number nodes that can be removed for networks trained on different architectures and datasets according to the amount of regularization used.

Runtime improvement Fig. 1 compares the baseline [74] with our approach on smaller MNIST classifiers—2×100, 2×200, and 2×400—using ℓ_1 as described above. The median ratio between runtimes is 100. The overall speedup is greater in larger networks: the median runtime ratio is 77 for 2×100, 153 for 2×200, and 193 for 2×400. By comparing the runtimes when not timing out with and without the preprocessing step in 2×100, we observed a median speed up of 3.2.

Effect of regularization on compressibility We observe more compression with more ℓ_1 regularization. For sufficiently large networks having the same accuracy as those trained with $\ell_1 = 0$ on MNIST, we can remove around 20% of the neurons and 40% of the connections. In line with [74], we observe that the exact compressibility of neural networks trained with $\ell_1 = 0$ is negligible, but also that you can have the cake and eat it too: certain choices of regularization lead to better accuracy and a smaller network.

Relationship between compressibility and accuracy Fig. 2 analyzes the relationship between classifier accuracy and the number of neurons left after compression for 2×100. When excluding uncompressible networks with $\ell_1 = 0$ or sufficiently small, we obtain a linear regression with coefficient of determination $R^2 = 0.64$. That suggests that the accuracy is a good proxy for how much a neural network trained with ℓ_1 regularization can be compressed.

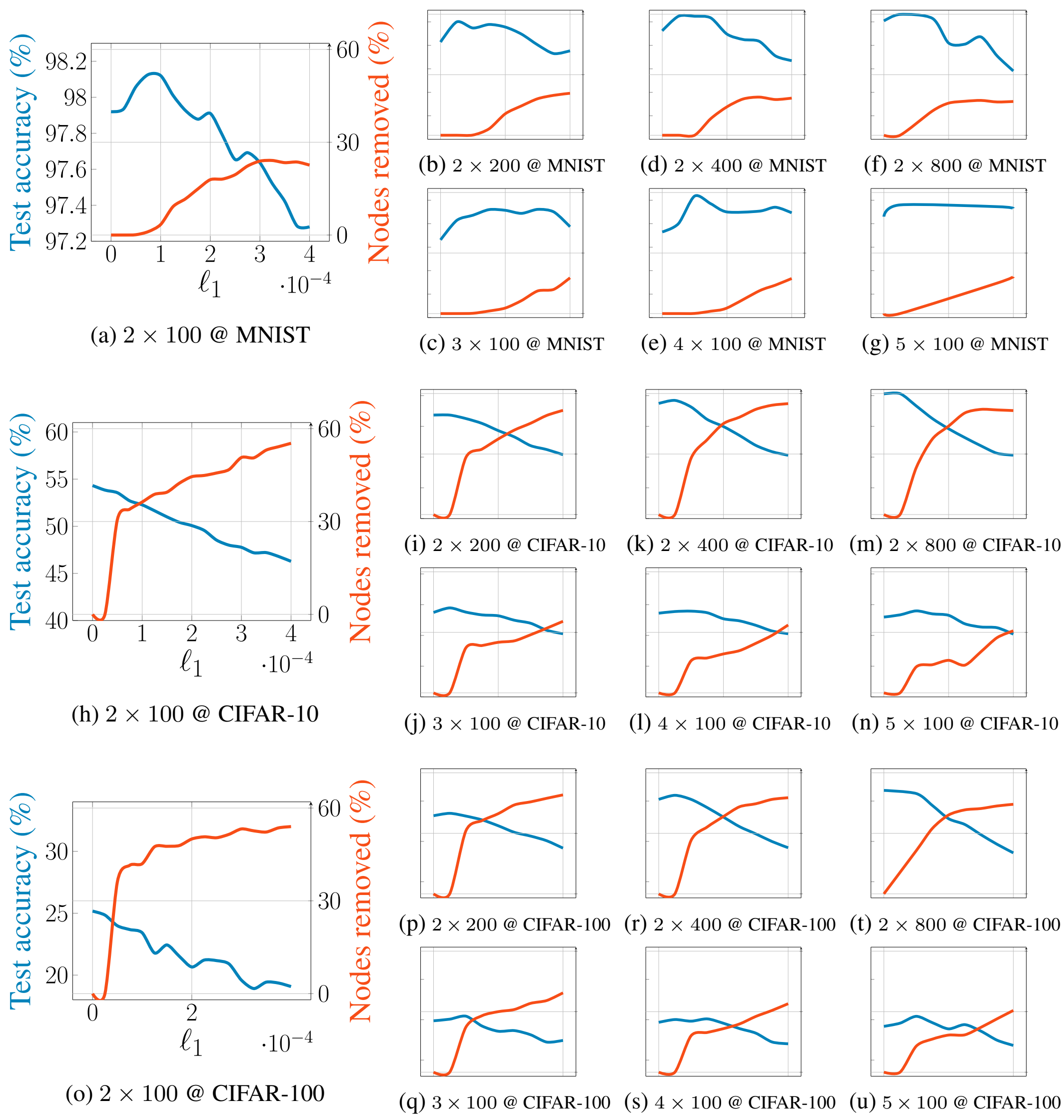


Figure 3: **Test accuracy and nodes removed for varying amounts of ℓ_1 regularization.** The plots correspond to classifiers with different architectures on the (a)-(g) MNIST, (h)-(n) CIFAR-10, and (o)-(u) CIFAR-100 datasets. For each dataset, we keep the ranges of all the axes of the smaller plots same as the bigger plot but hide the ticks for brevity. Networks trained with ℓ_1 regularization can be exactly compressed, even when regularization improves accuracy.

Limitations Due to the need to solve MILP formulations, our approach is not applicable to very large networks. In particular, non-exact compression methods would be more practical and easily applicable on larger networks, despite not providing any guarantees.

7 Conclusion

This paper outlined the potential for exact compression of neural networks and presented an approach that makes it practical for sizes that are large enough for many applications. To the best of our knowledge, our approach is the state-of-the-art for optimization-based exact compression.

Our performance improvements come from insights about the MILP formulations associated with optimization problems over neural networks, which have many other applications besides exact compression. Most notably, such formulations are also used for network verification [14; 56; 72].

Societal Impact Large models are resource-intensive for both training as well as inference. In contrast to approximate methods, our exact model compression algorithms can help deep learning practitioners to save computational time and resources without worrying about any loss in performance. That helps preventing the documented side effect of disproportionally degrading performance in minoritized classes—which often correspond to minoritized groups in society—when the indicator of a successfull compression is the overall performance.

References

[1] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg. Net-Trim: Convex pruning of deep neural networks with performance guarantee. In *NeurIPS*, 2017.

[2] A. Al-Falou and D. Trummer. Identifiability of recurrent neural networks. *Econometric Theory*, 2003.

[3] F. Albertini and E. Sontag. For neural networks, function determines form. *Neural Networks*, 1993.

[4] F. Albertini and E. Sontag. Identifiability of discrete-time neural networks. In *ECC*, 1993.

[5] D. Amodei, D. Hernandez, G. Sastry, J. Clark, G. Brockman, and I. Sutskever. AI and compute. `https://openai.com/blog/ai-and-compute/`, 2018. Accessed: 2020-12-23.

[6] R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

[7] R. Anderson, J. Huchette, C. Tjandraatmadja, and J. Vielma. Strong mixed-integer programming formulations for trained neural networks. In *IPCO*, 2019.

[8] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. Stronger generalization bounds for deep nets via a compression approach. In *ICML*, 2018.

[9] D. Bergman, T. Huang, P. Brooks, A. Lodi, and A. Raghunathan. JANOS: An integrated predictive and prescriptive modeling framework. *arXiv*, 1911.09461, 2019.

[10] J. Berner, D. Elbrächter, and P. Grohs. How degenerate is the parametrization of neural networks with the relu activation function? In *NeurIPS*, 2019.

[11] D. Blalock, J. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In *MLSys*, 2020.

[12] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI*, 2020.

[13] J. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*. 1990.

[14] R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and M. Pawan Kumar. Piecewise linear neural network verification: A comparative study. *CoRR*, abs/1711.00455, 2017.

[15] M. Büning, P. Kern, and C. Sinz. Verifying equivalence properties of neural networks with relu activation functions. In *CP*, 2020.

[16] A. Chen, H. Lu, and R. Hecht-Nielsen. On the geometry of feedforward neural network error surfaces. *Neural Computation*, 1993.

[17] C. Cheng, G. Nührenberg, and H. Ruess. Maximum resilience of artificial neural networks. In *ATVA*, 2017.

[18] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 2018.

[19] S. Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.

[20] G. Dantzig, D. Fulkerson, and S. Johnson. Solution of a large scale traveling salesman problem. Technical Report P-510, RAND Corporation, Santa Monica, California, USA, 1954.

[21] A. Delarue, R. Anderson, and C. Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. In *NeurIPS*, 2020.

[22] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. Freitas. Predicting parameters in deep learning. In *NeurIPS*, 2013.

[23] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NeurIPS*, 2014.

[24] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS*, 2017.

[25] M. ElAraby, G. Wolf, and M. Carvalho. Identifying efficient sub-networks using mixed integer programming. In *OPT Workshop*, 2020.

[26] C. Fefferman and S. Markel. Recovering a feed-forward net from its output. In *NeurIPS*, 1994.

[27] M. Fischetti and J. Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 2018.

[28] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[29] J. Frankle, G. Dziugaite, D. Roy, and M. Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *ICLR*, 2021.

[30] M. Gordon, K. Duh, and N. Andrews. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Rep4NLP Workshop*, 2020.

[31] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. Version 9.1.

[32] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas, and S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 2000.

[33] S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *ICLR*, 2016.

[34] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

[35] B. Hanin and D. Rolnick. Complexity of linear regions in deep networks. In *ICML*, 2019.

[36] B. Hanin and D. Rolnick. Deep ReLU networks have surprisingly few activation patterns. In *NeurIPS*, 2019.

[37] S. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. In *NeurIPS*, 1988.

[38] B. Hassibi, D. Stork, and G. Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, 1993.

[39] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[40] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[41] R. Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*. 1990.

[42] S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome. What do compressed deep neural networks forget? *arXiv*, 1911.05248, 2019.

[43] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[44] S. Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 1989.

[45] A. Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[46] V. Kůrková and P. Kainen. Functionally equivalent feedforward neural networks. *Neural Computation*, 1994.

[47] A. Kumar, T. Serra, and S. Ramalingam. Equivalent and approximate transformations of deep neural networks. *arXiv*, 1905.11428, 2019.

[48] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *ICLR*, 2015.

[49] V. Lebedev and V. Lempitsky. Fast ConvNets using group-wise brain damage. In *CVPR*, 2016.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[51] Y. LeCun, Denker, J., and S. Solla. Optimal brain damage. In *NeurIPS*, 1989.

[52] N. Lee, T. Ajanthan, S. Gould, and P. Torr. A signal propagation perspective for pruning neural networks at initialization. In *ICLR*, 2020.

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

[54] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[55] J. Li, Y. Sun, J. Su, T. Suzuki, and F. Huang. Understanding generalization in deep learning via tensor methods, 2020.

[56] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. Kochenderfer. Algorithms for verifying deep neural networks. *CoRR*, 2019.

[57] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[58] J. Luo, J. Wu, and W. Lin. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[59] Z. Mariet and S. Sra. Diversity networks: Neural network compression using determinantal point processes. In *ICLR*, 2016.

[60] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

[61] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NeurIPS*, 2014.

[62] M. Mozer and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1989.

[63] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying properties of binarized deep neural networks. In *AAAI*, 2018.

[64] R. Pascanu, G. Montúfar, and Y. Bengio. On the number of response regions of deep feedforward networks with piecewise linear activations. In *ICLR*, 2014.

[65] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[66] P. Petersen, M. Raslan, and F. Voigtlaender. Topological properties of the set of functions generated by neural networks of fixed size. *Foundations of Computational Mathematics*, 2020.

[67] M. Phuong and C. Lampert. Functional vs. parametric equivalence of ReLU networks. In *ICLR*, 2020.

[68] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Dickstein. On the expressive power of deep neural networks. In *ICML*, 2017.

[69] A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020.

[70] D. Rolnick and K. Kording. Reverse-engineering deep ReLU networks. In *ICML*, 2020.

[71] J. Rosenfeld, J. Frankle, M. Carbin, and N. Shavit. On the predictability of pruning across scales. *arXiv*, 2006.10621, 2020.

[72] A. Rössig and M. Petkovic. Advances in verification of ReLU neural networks. *Journal of Global Optimization*, 2020.

[73] B. Say, G. Wu, Y. Zhou, and S. Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *IJCAI*, 2017.

[74] T. Serra, A. Kumar, and S. Ramalingam. Lossless compression of deep neural networks. In *CPAIOR*, 2020.

[75] T. Serra and S. Ramalingam. Empirical bounds on linear regions of deep rectifier networks. In *AAAI*, 2020.

[76] T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and counting linear regions of deep neural networks. In *ICML*, 2018.

[77] G. Sourek and F. Zelezny. Lossless compression of structured convolutional models via lifting. In *ICLR*, 2021.

[78] S. Srinivas and R. Venkatesh Babu. Data-free parameter pruning for deep neural networks. In *BMVC*, 2015.

[79] J. Su, J. Li, B. Bhattacharjee, and F. Huang. Tensorial neural networks: Generalization of neural networks and application to model compression, 2018.

[80] X. Suau, L. Zappella, and N. Apostoloff. Filter distillation for network compression. In *WACV*, 2020.

[81] T. Suzuki, H. Abe, T. Murata, S. Horiuchi, K. Ito, T. Wachi, S. Hirai, M. Yukishima, and T. Nishimura. Spectral-pruning: Compressing deep neural network via spectral analysis. In *IJCAI*, 2020.

[82] T. Suzuki, H. Abe, and T. Nishimura. Compression based bound for non-compressed network: Unified generalization error analysis of large compressible deep neural network. In *ICLR*, 2020.

[83] H. Tanaka, D. Kunin, D. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020.

[84] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *ICLR*, 2019.

[85] C. Wang, R. Grosse, S. Fidler, and G. Zhang. EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In *ICML*, 2019.

[86] C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

[87] Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets, 2018.

[88] K. Xiao, V. Tjeng, N. Shafiullah, and A. Madry. Training for faster adversarial robustness verification via inducing ReLU stability. *ICLR*, 2019.

[89] X. Xing, L. Sha, P. Hong, Z. Shang, and J. Liu. Probabilistic connection importance inference and lossless compression of deep neural networks. In *ICLR*, 2020.

[90] R. Yu, A. Li, C. Chen, J. Lai, V. Morariu, X. Han, M. Gao, C. Lin, and L. Davis. NISP: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[91] W. Zeng and R. Urtasun. MLPrune: Multi-layer pruning for automated neural network compression. 2018.

[92] W. Zhou, V. Veitch, M. Austern, R. Adams, and P. Orbanz. Non-vacuous generalization bounds at the ImageNet scale: A PAC-Bayesian compression approach. In *ICLR*, 2019.

Scaling Up Exact Neural Network Compression by ReLU Stability

Supplementary Material

A Description of MILP formulation for ReLU activation

$$\boldsymbol{w}_i^l \cdot \boldsymbol{x}^{l-1} + b_i^l = y_i^l \tag{13}$$

$$y_i^l = x_i^l - \chi_i^l \tag{14}$$

$$x_i^l \leq M_i^l z_i^l \tag{15}$$

$$\chi_i^l \leq \mu_i^l (1 - z_i^l) \tag{16}$$

$$x_i^l \geq 0 \tag{17}$$

$$\chi_i^l \geq 0 \tag{18}$$

$$z_i^l \in \{0, 1\} \tag{19}$$

Constraint (13) matches the layer input $\boldsymbol{x}^{l-1}$ with the neuron preactivation output y_i^l. We then use the binary variable z_i^l to match y_i^l with the neuron output with either x_i^l or 0. When $z_i^l = 1$, constraints (16) and (18) imply that $\chi_i^l = 0$, and thus $x_i^l = y_i^l$ due to constraint (14). That only happens if $y_i^l \geq 0$ due to constraint (17). When $z_i^l = 0$, constraints (15) and (17) imply that $x_i^l = 0$, and thus $\chi_i^l = -y_i^l$. That only happens if $y_i^l \leq 0$ due to constraint (18).

B On dropping constraint (12)

We avoid explicitly enforcing that variables p_i^l and q_i^l are binary by leveraging that z_i^l is binary. Constraint (10) implies that $p_i^l \in [0, 1]$ and $p_i^l \neq 0$ only if $z_i^l = 1$. In turn, if $z_i^l = 1$, then we can assume $p_i^l = 1$ by optimality since the objective function (7) maximizes the sum of those variables and no other constraint limits its value. Likewise, constraint (11) implies that $q_i^l \in [0, 1]$ and $q_i^l \neq 0$ only if $z_i^l = 0$. In turn, if $z_i^l = 0$, then likewise we can assume $q_i^l = 1$ by optimality since the objective function (7) maximizes the sum of those variables and no other constraint limits its value. Reducing the number of binary variables is widely regarded as a good practice to make MILP formulations easier to solve.

C Proofs from Section 5.1

Proposition 1. *If $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = 0$, then every neuron $i \in \mathbb{P}^l$ is stably inactive and every neuron $i \in \mathbb{Q}^l$ is stably active.*

Proof. Constraint (10) is the only upper bound on p_i^l besides constraint (12). Hence, if there is any solution $(\bar{x}, \bar{z}, \bar{p}, \bar{q})$ of (9)–(12) in which $\bar{z}_i^l = 1$ for some $i \in \mathbb{P}_i^l, l \in \mathbb{L}$, then either $\bar{p}_i^l = 1$ or there is another solution $(\bar{\bar{x}}, \bar{\bar{z}}, \bar{\bar{p}}, \bar{\bar{q}})$ in which $\bar{\bar{p}}_i^l = 1$ and all other variables have the same value.

Likewise, constraint (10) is the only upper bound on q_i^l besides constraint (12). Hence, if there is any solution $(\bar{x}, \bar{z}, \bar{p}, \bar{q})$ of (9)–(12) in which $\bar{z}_i^l = 0$ for some $i \in \mathbb{P}_i^l, l \in \mathbb{L}$, then either $\bar{q}_i^l = 1$ or there is another solution $(\bar{\bar{x}}, \bar{\bar{z}}, \bar{\bar{p}}, \bar{\bar{q}})$ in which $\bar{\bar{q}}_i^l = 1$ and all other variables have the same value.

If $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = 0$, then for every solution $(\bar{x}, \bar{z}, \bar{p}, \bar{q})$ it follows that $\bar{p}_i^l = 0 \ \forall i \in \mathbb{P}^l, l \in \mathbb{L}$ and $\bar{q}_i^l = 0 \ \forall i \in \mathbb{Q}^l, l \in \mathbb{L}$, and consequently $\bar{z}_i^l = 0 \ \forall i \in \mathbb{P}^l, l \in \mathbb{L}$ and $\bar{z}_i^l = 1 \ \forall i \in \mathbb{Q}^l, l \in \mathbb{L}$. Thus, the neurons in $\mathbb{P}^l$ are always inactive and the neurons in $\mathbb{Q}^l$ are always active for any valid input. □

Corollary 2. *The stability of all neurons of a neural network can be determined by solving formulation* (7)–(12) *at most $N + 1$ times, where $N := \sum_{l \in \mathbb{L}} n_l$.*

Proof. Let us initially consider a formulation in which $\mathbb{P}^l = \mathbb{Q}^l = \{1, \ldots, n_l\}\ \forall l \in \mathbb{L}$ and then respectively remove from those sets each neuron i for which $p_i^l = 1$ and $q_i^l = 1$ in any solution obtained. When the formulation is first solved, we remove each neuron from either $\mathbb{P}^l$ or $\mathbb{Q}^l$, and therefore N states remain unobserved. In subsequent steps, either (i) $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) > 0$ and the number of unobserved states decreases; or (ii) $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q}) = 0$, and thus any neuron $i \in \mathbb{P}^l$ is stably inactive and any neuron $i \in \mathbb{Q}^l$ is stably active. □

D On lazy constraint callbacks

Lazy constraint callbacks are generally used when the total number of constraints of an MILP formulation is prohibitively large. One such example is the most commonly used formulation for the traveling salesperson problem due to the subtour elimination constraints [20]. The callback allows us to handle such cases more efficiently by formulating the problem with fewer constraints and then adding the remaining ones only if they are necessary to rule out infeasible solutions. Every time that a supposedly feasible solution is found, the MILP solver invokes the callback implemented by the user for an opportunity to make such a solution infeasible by adding one of the missing constraints that the supposedly feasible solution does not satisfy. If none is provided by the callback, the MILP solver accepts the solution as feasible.

In our case, we use a lazy constraint callback for a slightly different purpose. Namely, we implement the callback to (i) inspect every feasible solution that is obtained; and (ii) mimic the updates that would have been made to $\boldsymbol{P}$ and $\boldsymbol{Q}$ between consecutive calls to the solver by adding constraints that set the value of either p_i^l or q_i^l to zero once a solution is found in which such variable has a positive value. In other words, the callback adds constraints to ignore the effect of p_i^l or q_i^l on the objective function if we know that the i-th neuron of layer l is active or inactive for some input, respectively. Therefore, the MILP solver will eventually produce an optimal solution of value zero once the set of solutions inspected by the callback covers all the possible states for the neurons and the remaining states are deemed unattainable after an exhaustive search.

E Algorithm for identifying stable neurons

Algorithm 2 identifies all stable neurons of a neural network. The prior discussion on identifying stable units leads to the steps described between lines 3 and 19. First, $\boldsymbol{P}$ and $\boldsymbol{Q}$ are initialized between lines 3 and 5. Next, the MILP formulation is iteratively solved between lines 6 and 19. The block between lines 7 and 8 identifies the termination criterion, which implies that the unobserved states cannot be obtained with any valid input. The block between lines 9 and 15 inspects every feasible solution to identify unobserved states and then to effectively remove the decision variables associated with those states from the objective function by adding a constraint that sets their value to 0. The block between lines 16 and 17 produces a feasible solution from a solution of the LP relaxation when the latter is produced by the MILP solver. For brevity, we assume that the block between lines 9 and 15 would leverage such solution at the next repetition of the loop.

F Proofs from Section 5.4

Proposition 3. *Let $\mathbb{S}$ be a set of stably active neurons in layer l. If $r := \text{rank}(\boldsymbol{W}_S^l) < |S|$ and let $\mathbb{T} \subset \mathbb{S}$ be a subset of those neurons for which $\text{rank}(\boldsymbol{W}_{\mathbb{T}}^l) = r$, then the output of the neurons in $\mathbb{S} \setminus \mathbb{T}$ is an affine function on the output of the neurons in $\mathbb{T}$.*

Proof. For every $i \in \mathbb{S} \setminus \mathbb{T}$, there is a vector $\boldsymbol{\alpha}^i \in \mathbb{R}^r$ such that $\boldsymbol{w}_i^l = \sum_{j \in \mathbb{T}} \alpha_j^i \boldsymbol{w}_j^l$. Since $\boldsymbol{x}_i^l = \boldsymbol{w}_i^l \cdot \boldsymbol{x}^{l-1} + b_i^l$ for every $i \in \mathbb{S}$ because all neurons in $\mathbb{S}$ are stably active, then for every $i \in \mathbb{S} \setminus \mathbb{T}$ it follows that $\boldsymbol{x}_i^l = \sum_{j \in \mathbb{T}} \alpha_j^i \boldsymbol{w}_j^l \cdot \boldsymbol{x}^{l-1} + b_i^l = \sum_{j \in \mathbb{T}} \alpha_j^i \left(\boldsymbol{w}_j^l \cdot \boldsymbol{x}^{l-1} + b_j^l\right) + \left(b_i^l - \sum_{j \in \mathbb{T}} \alpha_j^i b_j^l\right) = \sum_{j \in \mathbb{T}} \alpha_j^i x_j^l + \left(b_i^l - \sum_{j \in \mathbb{T}} \alpha_j^i b_j^l\right)$. □

Corollary 4. *If $\mathbb{S}$, $\mathbb{T}$, and l are such as in Proposition 3, then the pre-activation output of the neurons in layer $l + 1$ is an affine function on the outputs of all neurons from layer l with exception of the neurons in $\mathbb{T}$.*

Algorithm 2 Identifies all stable neurons of the neural network

Input: neural network $\left(L, \left\{(n_l, \boldsymbol{W}^l, \boldsymbol{b}^l)\right\}_{l \in \mathbb{L}}\right)$
Output: stable neurons $\left(\left\{(\mathbb{P}^l, \mathbb{Q}^l)\right\}_{l \in \mathbb{L}}\right)$
for $l \leftarrow 1$ **to** L **do**
  $\mathbb{P}^l \leftarrow \mathbb{Q}^l = \{1, \ldots, n_l\}$
end for
while solving $\mathcal{C}(\boldsymbol{P}, \boldsymbol{Q})$ **do**
  if optimal value is proven to be 0 **then**
    break
  else if found positive MILP solution $(\bar{x}, \bar{z}, \bar{p}, \bar{q})$ **then**
    for $l \leftarrow 1$ **to** L **do**
      $\mathbb{P}^l \leftarrow \mathbb{P}^l \setminus \{i : i \in \mathbb{P}^l$ and $\bar{p}_i^l > 0\}$
      set $p_i^l = 0$ for every $i \in \mathbb{P}^l$ such that $\bar{p}_i^l > 0$
      $\mathbb{Q}^l \leftarrow \mathbb{Q}^l \setminus \{i : i \in \mathbb{Q}^l$ and $\bar{q}_i^l > 0\}$
      set $q_i^l = 0$ for every $i \in \mathbb{Q}^l$ such that $\bar{q}_i^l > 0$
    end for
  else if found LP relaxation solution $(\tilde{x}, \tilde{z}, \tilde{p}, \tilde{q})$ **then**
    use $\tilde{x}^0$ to produce an MILP solution $(\bar{x}, \bar{z}, \bar{p}, \bar{q})$
  end if
end while
return $\left(\left\{(\mathbb{P}^l, \mathbb{Q}^l)\right\}_{l \in \mathbb{L}}\right)$

Proof. Let $\mathbb{U} := \{1, \ldots, n_l\} \setminus \mathbb{S}$. The pre-activation output of every neuron i in layer $l+1$ is given by $y_i^{l+1} = \sum_{j \in \mathbb{U} \cup \mathbb{S}} w_{ij}^{l+1} x_j^l + b_i^{l+1} = \sum_{j \in \mathbb{U} \cup \mathbb{T}} w_{ij}^{l+1} x_j^l + \sum_{j \in \mathbb{S} \setminus \mathbb{T}} w_{ij}^{l+1} \left(\sum_{k \in \mathbb{T}} \alpha_k^j x_k^l + \left(b_j^l - \sum_{k \in \mathbb{T}} \alpha_k^j b_k^l \right) \right) + b_i^{l+1} = \sum_{j \in \mathbb{U}} w_{ij}^{l+1} x_j^l + \sum_{j \in \mathbb{T}} \left(w_{ij}^{l+1} + \sum_{k \in \mathbb{S} \setminus \mathbb{T}} \alpha_j^k w_i^{l+1} k \right) x_j^l + \left(b_i^{l+1} + \sum_{j \in \mathbb{S} \setminus \mathbb{T}} w_{ij}^{l+1} \left(b_j^l - \sum_{k \in \mathbb{T}} \alpha_k^j b_k^l \right) \right)$. □

Proposition 5. *If all the neurons of layer $l \in \mathbb{L} \setminus \{L\}$ are stably active, then the pre-activation output of layer $l+1$ is an affine function on the inputs of layer l.*

Proof. Since $\boldsymbol{x}^l = \boldsymbol{W}^l \boldsymbol{x}^{l-1} + \boldsymbol{b}^l$, then $\boldsymbol{y}^{l+1} = \boldsymbol{W}^{l+1} \boldsymbol{x}^l + \boldsymbol{b}^{l+1} = \boldsymbol{W}^{l+1} \boldsymbol{W}^l \boldsymbol{x}^{l-1} + \left(\boldsymbol{W}^{l+1} \boldsymbol{b}^l + \boldsymbol{b}^{l+1} \right)$. □

Corollary 6. *If all neurons of layer $l \in \mathbb{L} \setminus \{L\}$ are stable, then the pre-activation output of layer $l+1$ is an affine function on the inputs of layer l.*

Proof. Let $\mathbb{S}$ be the set of stably active neurons in layer l. If $|\mathbb{S}| < n_l$, the identity $\boldsymbol{x}^l = \boldsymbol{W}^l \boldsymbol{x}^{l-1} + \boldsymbol{b}^l$ still holds if the bias and the weights of all the connections of the neurons not in $\mathbb{S}$ with the neurons in the next layer are 0. More generally, we can thus obtain an equivalent neural network if $\boldsymbol{W}^l$ and $\boldsymbol{b}^l$ are both premultiplied by $\boldsymbol{I}_{n_l}(\mathbb{S})$ since that only would change the weights and biases associated with the neurons not in $\mathbb{S}$ to 0. Hence, the identity $\boldsymbol{x}^l = \boldsymbol{I}_{n_l}(\mathbb{S}) \left(\boldsymbol{W}^l \boldsymbol{x}^{l-1} + \boldsymbol{b}^l \right)$ always holds if all neurons in layer l are stable, which implies that $\boldsymbol{y}^{l+1} = \boldsymbol{W}^{l+1} \boldsymbol{I}_{n_l}(\mathbb{S}) \boldsymbol{W}^l \boldsymbol{x}^{l-1} + \left(\boldsymbol{W}^{l+1} \boldsymbol{I}_{n_l}(\mathbb{S}) \boldsymbol{b}^l + \boldsymbol{b}^{l+1} \right)$. □

G Implementation details

We now provide additional experimental results evaluating our proposed method and the baseline.

Architecture and Loss We implemented the fully connected architectures in PyTorch [65]. All the networks have ReLU activations but have varying number of layers and width. For the classifiers, we pass the output through a softmax layer and use negative log-likelihood loss as the loss function. For the autoencoders, we use MSE loss as the loss function.

Datasets and Splits We keep the output units at 10 and 784 for the MNIST dataset [50] classifiers and autoencoders, respectively. We keep the output units at 10 and 100 for the CIFAR-10 and the CIFAR-100 dataset [45] classifiers, respectively. We use the standard train-validation data splits of each of the datasets available in PyTorch.

Data Augmentation We do not do any data augmentation of training images of the MNIST dataset as in [74] for a fair comparison. We carry out the standard data augmentation of training images of the CIFAR-10 and CIFAR-100 datasets: horizontal flipping with probability 0.5, random rotation in the range between (-10^o, 10^o), random scaling in the range (0.8, 1.2), random shear parallel to the x axis in the range (-10, 10), and scaling the brightness, contrast, saturation and hue by a random factor in the range (0.8, 1.2).

Optimization Training proceeds from scratch for 120 epochs and starts with learning rate of 0.01, which is decayed by a factor of 0.1 after every 50 epochs as in [74]. We use SGD with momentum optimizer, with a momentum of 0.9 and batch size 128 as in [74]. Unless stated otherwise, we use ℓ_1 regularization. The weight decay is kept at 0 unless otherwise stated. We consider the model saved in the last epoch as our final model.

MILP Solver We solve the MILP formulations using Gurobi 9.1.0 through gurobipy [31]. We calculate the value of the positive constants M_i^l and μ_i^l for each neuron with an upper bound of on the values of x_i^l and χ_i^l through interval arithmetic by taking element-wise maxima [17].

Initialization We initialize the weights of the network with the Kaiming initialization [39] and the biases to zero with different random seeds for each training. We train every setting 5 times, and get the stably active and inactive neurons with the proposed approach to prune the network for each run. We omit from the summaries the runs which resulted in a time out. We keep the timeout to 3 hours.

Hardware The classifier experiments were run on a machine with Intel(R) Core(R) i7-4790 CPU @ 3.60 GHz and 32 GB of RAM, and one 4GB GeForce GTX 970 GPU. The autoencoder experiments were run on a machine with 40 Intel(R) Xeon(R) E5-2640 CPU @ 2.40GHz processors, 126 GB of RAM, and one 12GB Nvidia Titan Xp GPU.

H Additional experiments and results

H.1 MNIST Classifiers

Relationship between Runtime and Regularization Tab. 1 and Tab. 2 show the runtime achieved by the proposed method at different ℓ_1 regularization on MNIST classifiers.

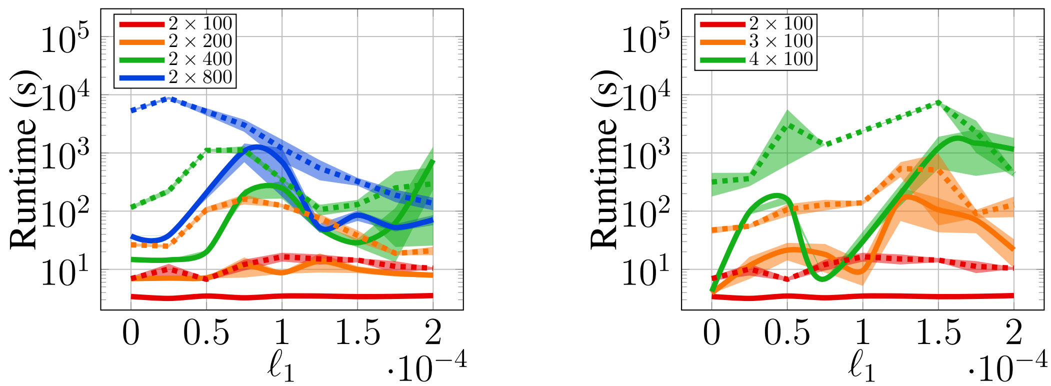
Table 1: **MNIST Classifiers:** Compression results with fixed width and varying depth.

Arch.	ℓ_1	Accuracy (%)	Compression Runtime (s)	% Removed Neurons	% Removed Connections	Timed Out
2×100	0	97.92 ± 0.09	3.4 ± 0.3	0 ± 0	0 ± 0	0
2×100	0.000025	97.93 ± 0.02	3.2 ± 0.1	0 ± 0	0 ± 0	0
2×100	0.00005	98.06 ± 0.09	3.5 ± 0.3	0.1 ± 0.2	0.2 ± 0.4	0
2×100	0.000075	98.13 ± 0.09	3.2 ± 0.2	1.1 ± 0.4	2 ± 0.8	0
2×100	0.0001	98.12 ± 0.09	3.5 ± 0.1	3.4 ± 0.7	6 ± 1	0
2×100	0.000125	98.01 ± 0.09	3.5 ± 0.3	9.2 ± 0.6	17 ± 1	0
2×100	0.00015	97.9 ± 0.1	3.4 ± 0.3	12 ± 2	21 ± 4	0
2×100	0.000175	97.88 ± 0.05	3.4 ± 0.3	15 ± 3	26 ± 4	0
2×100	0.0002	97.91 ± 0.1	3.5 ± 0.4	18 ± 2	31 ± 3	0
2×100	0.000225	97.8 ± 0.1	4.2 ± 0.9	18 ± 3	31 ± 5	0
2×100	0.00025	97.65 ± 0.09	4 ± 0.5	20 ± 2	34 ± 4	0
2×100	0.000275	97.69 ± 0.09	4 ± 1	22 ± 2	38 ± 3	0
2×100	0.0003	97.64 ± 0.06	3.8 ± 0.4	24 ± 2	40 ± 4	0
2×100	0.000325	97.52 ± 0.08	3.5 ± 0.3	24 ± 3	41 ± 4	0
2×100	0.00035	97.42 ± 0.04	4 ± 1	23 ± 3	39 ± 4	0
2×100	0.000375	97.3 ± 0.2	3.4 ± 0.3	24 ± 3	40 ± 5	0
2×100	0.0004	97.28 ± 0.03	4.1 ± 0.7	23 ± 2	38 ± 3	0
3×100	0	97.86 ± 0.06	3.9 ± 0.1	0 ± 0	0 ± 0	0
3×100	0.000025	98.03 ± 0.08	10 ± 10	0 ± 0	0 ± 0	0
3×100	0.00005	98.1 ± 0.1	20 ± 10	0.1 ± 0.3	0.2 ± 0.4	0
3×100	0.000075	98.12 ± 0.07	20 ± 20	1.3 ± 0.7	1.8 ± 1	0
3×100	0.0001	98.11 ± 0.09	8 ± 8	2.7 ± 0.9	4 ± 1	0
3×100	0.000125	98.09 ± 0.1	2000 ± 4000	6 ± 1	11 ± 3	0
3×100	0.00015	98.1 ± 0.1	100 ± 100	11 ± 2	20 ± 3	0
3×100	0.000175	98.1 ± 0.1	70 ± 60	12 ± 2	20 ± 2	0
3×100	0.0002	98 ± 0.1	20 ± 20	18 ± 2	30 ± 3	0
4×100	0	97.93 ± 0.07	4.2 ± 0.2	0 ± 0	0 ± 0	0
4×100	0.000025	98 ± 0.1	200 ± 200	0 ± 0	0 ± 0	0
4×100	0.00005	98.23 ± 0.08	1000 ± 3000	0.1 ± 0.1	0.1 ± 0.2	1
4×100	0.000075	98.17 ± 0.09	1000 ± 1000	1.2 ± 0.4	1.5 ± 0.5	2
4×100	0.0001	98.1 ± 0.06	3000 ± 3000	2.8 ± 0.9	4 ± 1	2
4×100	0.00015	98.1 ± 0.2	2000 ± 1000	11 ± 2	20 ± 4	2
4×100	0.000175	98.1 ± 0.1	1000 ± 2000	14 ± 1	24 ± 3	0
4×100	0.0002	98.09 ± 0.07	1000 ± 1000	17 ± 2	30 ± 3	1
5×100	0	98.06 ± 0.03	2000 ± 3000	0 ± 0	0 ± 0	1
5×100	0.000025	98.2 ± 0.1	1000 ± 100	0 ± 0	0 ± 0	3
5×100	0.000175	98.1 ± 0.2	4000 ± 4000	15.1 ± 0.7	27 ± 2	3
5×100	0.0002	98.1 ± 0.1	3000 ± 2000	18 ± 1	32 ± 2	1

Runtime Comparison with SoTA Fig. 4 shows the comparison of runtimes with the proposed method and the baseline with the strength of ℓ_1 regularization on the MNIST classifiers. We observe that the new method presents a median gain of **3.76** times in performance.

Table 2: **MNIST Classifiers:** Compression results with fixed height and varying width.

ARCHITECTURE	ℓ_1	ACCURACY (%)	COMPRESSION RUNTIME (S)	% REMOVED NEURONS	% REMOVED CONNECTIONS	TIMED OUT
2×100	0	97.92 ± 0.09	3.4 ± 0.3	0 ± 0	0 ± 0	0
2×100	0.000025	97.93 ± 0.02	3.2 ± 0.1	0 ± 0	0 ± 0	0
2×100	0.00005	98.06 ± 0.09	3.5 ± 0.3	0.1 ± 0.2	0.2 ± 0.4	0
2×100	0.000075	98.13 ± 0.09	3.2 ± 0.2	1.1 ± 0.4	2 ± 0.8	0
2×100	0.0001	98.12 ± 0.09	3.5 ± 0.1	3.4 ± 0.7	6 ± 1	0
2×100	0.000125	98.01 ± 0.09	3.5 ± 0.3	9.2 ± 0.6	17 ± 1	0
2×100	0.00015	97.9 ± 0.1	3.4 ± 0.3	12 ± 2	21 ± 4	0
2×100	0.000175	97.88 ± 0.05	3.4 ± 0.3	15 ± 3	26 ± 4	0
2×100	0.0002	97.91 ± 0.1	3.5 ± 0.4	18 ± 2	31 ± 3	0
2×100	0.000225	97.8 ± 0.1	4.2 ± 0.9	18 ± 3	31 ± 5	0
2×100	0.00025	97.65 ± 0.09	4 ± 0.5	20 ± 2	34 ± 4	0
2×100	0.000275	97.69 ± 0.09	4 ± 1	22 ± 2	38 ± 3	0
2×100	0.0003	97.64 ± 0.06	3.8 ± 0.4	24 ± 2	40 ± 4	0
2×100	0.000325	97.52 ± 0.08	3.5 ± 0.3	24 ± 3	41 ± 4	0
2×100	0.00035	97.42 ± 0.04	4 ± 1	23 ± 3	39 ± 4	0
2×100	0.000375	97.3 ± 0.2	3.4 ± 0.3	24 ± 3	40 ± 5	0
2×100	0.0004	97.28 ± 0.03	4.1 ± 0.7	23 ± 2	38 ± 3	0
2×200	0	98.03 ± 0.05	6.9 ± 0.7	0 ± 0	0 ± 0	0
2×200	0.000025	98.2 ± 0.05	7.1 ± 0.7	0 ± 0	0 ± 0	0
2×200	0.00005	98.15 ± 0.04	7.2 ± 0.4	0.1 ± 0.1	0.2 ± 0.3	0
2×200	0.000075	98.18 ± 0.09	12 ± 9	3 ± 1	6 ± 2	0
2×200	0.0001	98.16 ± 0.07	8.8 ± 0.7	11 ± 1	20 ± 2	0
2×200	0.000125	98.1 ± 0.09	14 ± 10	15 ± 2	26 ± 3	0
2×200	0.00015	98 ± 0.02	10 ± 3	18 ± 2	32 ± 3	0
2×200	0.000175	97.9 ± 0.1	9 ± 2	20 ± 2	35 ± 3	0
2×200	0.0002	97.95 ± 0.08	8 ± 2	20.8 ± 0.6	36.6 ± 1	0
2×400	0	98.1 ± 0.1	14.8 ± 0.4	0 ± 0	0 ± 0	0
2×400	0.000025	98.25 ± 0.09	14.5 ± 0.5	0 ± 0	0 ± 0	0
2×400	0.00005	98.25 ± 0.07	20 ± 2	0 ± 0	0 ± 0	0
2×400	0.000075	98.23 ± 0.07	180 ± 80	8 ± 1	16 ± 2	0
2×400	0.0001	98.1 ± 0.09	200 ± 100	14 ± 1	26 ± 2	0
2×400	0.000125	98.05 ± 0.08	50 ± 20	18 ± 1	32 ± 2	0
2×400	0.00015	98.03 ± 0.05	29 ± 10	19 ± 2	34 ± 3	0
2×400	0.000175	97.9 ± 0.1	100 ± 100	17.7 ± 0.8	32 ± 1	0
2×400	0.0002	97.87 ± 0.1	1000 ± 1000	18 ± 1	33 ± 2	0
2×800	0	98.21 ± 0.05	37.6 ± 0.3	0 ± 0	0 ± 0	0
2×800	0.000025	98.26 ± 0.05	38.2 ± 0.4	0 ± 0	0 ± 0	0
2×800	0.000075	98.23 ± 0.03	1300 ± 800	12 ± 0.7	22 ± 1	0
2×800	0.0001	98 ± 0.1	1000 ± 1000	15.9 ± 0.9	29 ± 1	0
2×800	0.000125	98.01 ± 0.07	100 ± 100	16.8 ± 0.8	31 ± 1	0
2×800	0.00015	98.07 ± 0.06	90 ± 30	17.3 ± 0.6	31 ± 1	0
2×800	0.000175	97.91 ± 0.07	50 ± 20	16.5 ± 0.9	30 ± 2	0
2×800	0.0002	97.78 ± 0.06	80 ± 30	16.7 ± 0.6	31 ± 1	0



(a) With width

(b) With depth

Figure 4: **MNIST Classifiers: Comparison of runtimes** for proposed method (solid) and baseline (dashed) with the strength of regularization to identify stable neurons: (a) with increasing width (b) with increasing depth. We report the average and the standard deviation of the runtime of models with five different initialization for each regularization. Note that the y-axis is in the log scale. The median speedup is **3.76** times.

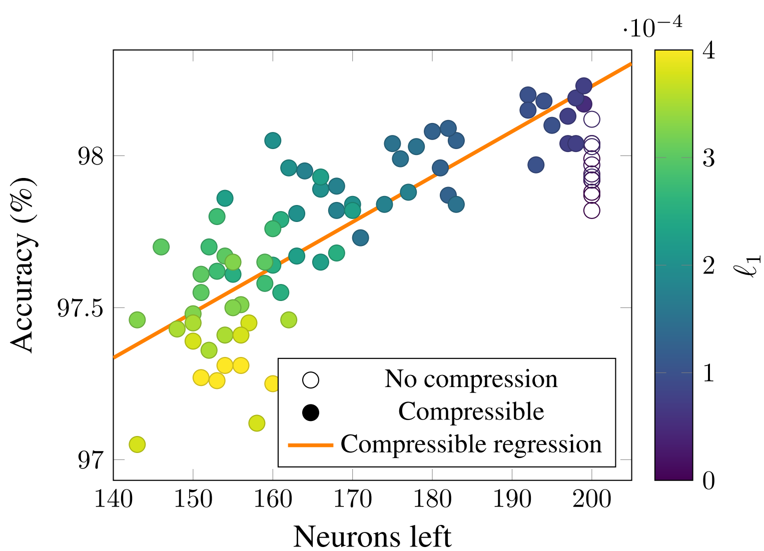


Figure 5: **Relationship between size of compressed neural network and accuracy on** 2×100 **MNIST classifiers.** The coefficient of determination (R^2) for linear regression obtained on the relationship between accuracy and neurons left for compressible networks is 61%.

H.2 MNIST Autoencoders

For the autoencoders, we use the notation $n_1 \mid n_2 \mid n_3$ for the architecture of 3 hidden layers with n_1, n_2, and n_3 neurons. The output layer has the same size as the input, 784, and uses ReLU activation. Starting with the architecture 100 | 10 | 100, we evaluated changes to the bottleneck width n_2 as well as to the width of the other two layers. First, we changed the bottleneck width to $n_2 = 25$ and $n_2 = 50$. Second, we changed the width of the other layers to $n_1, n_3 = 50$, $n_1, n_3 = 200$, and $n_1, n_3 = 400$ while keeping $n_2 = 10$. For each architecture, we trained and evaluated neural networks with 5 different random initialization seeds using $\ell_1 = 0$, $\ell_1 = 0.00002$, and $\ell_1 = 0.0002$.

Relationship between Runtime and Regularization Tab. 3 reports the runtime to identify stable neurons and the proportion of neurons—as well as the corresponding connections—that can be removed due to stability in each case on MNIST Autoencoders.

With the largest amount of regularization, we notice that the runtimes are considerably smaller and most of the network can be removed while the loss during training only doubles in comparison to using zero or a moderate amount of regularization. In fact, the only neurons that are not stable in such case are in the first layer, whereas between 3 and 4 out of the 5 neural networks trained for each architecture have all hidden layers completely stable. By also evaluating the stability of the output layer, we identified a few cases in which the output layer is entirely stable. While we have not explicitly explored that possibility in the proposed algorithm, the implication for such case is that the neural network can be reduced to a linear function on the domain of interest. With autoencoders, we observed that this can happen when the regularization during training no more than doubles the loss, and that we can evaluate if that happens within seconds: the runtime when the stability of the output layer is tested is 12 seconds on average and never more than 25 seconds.

Runtime Comparison with SoTA Fig. 6 shows the difference in runtimes between our approach and the baseline [74] for higher regularization, fixed $n_2 = 10$, and varying but equal values for n_1 and n_3 on the MNIST Autoencoders. We observe that the new method presents a median gain of **158.77** times in performance, which increases with the width of the non-bottleneck layers.

Table 3: **MNIST Autoencoders:** Compression results with varying architectures and levels of regularization.

Architecture	ℓ_1	Loss	Compression Runtime (s)	% Removed Neurons	% Removed Connections	Timed Out
100 | 10 | 100	0	0.045 ± 0.001	130 ± 30	0.1 ± 0.1	0.05 ± 0.06	0
100 | 10 | 100	0.00002	0.047 ± 0.0009	120 ± 30	12.7 ± 0.6	7.2 ± 0.9	0
100 | 10 | 100	0.0002	0.077 ± 0.002	2.73 ± 0.05	95 ± 6	90 ± 10	0
100 | 25 | 100	0	0.035 ± 0.001	500 ± 300	0 ± 0	0 ± 0	0
100 | 25 | 100	0.00002	0.047 ± 0.001	800 ± 200	14 ± 1	10 ± 2	0
100 | 25 | 100	0.0002	0.076 ± 0.001	2.88 ± 0.08	90 ± 7	80 ± 20	0
100 | 50 | 100	0	0.0311 ± 0.0009	230 ± 20	0 ± 0	0 ± 0	0
100 | 50 | 100	0.00002	0.0478 ± 0.0009	600 ± 200	17.4 ± 0.9	13 ± 1	0
100 | 50 | 100	0.0002	0.081 ± 0.003	2.96 ± 0.04	90 ± 7	80 ± 20	0
50 | 10 | 50	0	0.047 ± 0.002	33 ± 4	0 ± 0	0 ± 0	0
50 | 10 | 50	0.00002	0.051 ± 0.002	50 ± 20	14 ± 3	13 ± 2	0
50 | 10 | 50	0.0002	0.081 ± 0.002	1.42 ± 0.02	89 ± 8	88 ± 8	0
100 | 10 | 100	0	0.045 ± 0.001	130 ± 30	0.1 ± 0.1	0.05 ± 0.06	0
100 | 10 | 100	0.00002	0.047 ± 0.0009	120 ± 30	12.7 ± 0.6	7.2 ± 0.9	0
100 | 10 | 100	0.0002	0.077 ± 0.002	2.73 ± 0.05	95 ± 6	90 ± 10	0
200 | 10 | 200	0	0.041 ± 0.002	1000 ± 1000	0.4 ± 0.4	0.4 ± 0.4	1
200 | 10 | 200	0.00002	0.043 ± 0.002	700 ± 400	14 ± 0.7	7 ± 1	0
200 | 10 | 200	0.0002	0.076 ± 0.002	5.41 ± 0.03	95 ± 6	80 ± 20	0
400 | 10 | 400	0	0.04	2704	0	0	4
400 | 10 | 400	0.00002	0.0395 ± 0.001	1300 ± 100	15 ± 1	6 ± 0.7	0
400 | 10 | 400	0.0002	0.073 ± 0.001	10.5 ± 0.2	89.1 ± 7.5	13.6 ± 59.3	0

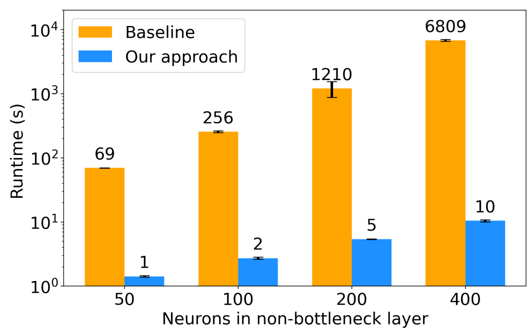


Figure 6: **MNIST Autoencoders: Comparison of runtimes** (in seconds) to identify stable neurons between the proposed approach vs. the baseline from [74] with high regularization ($\ell_1 = 0.0002$). Note that the y-axis is in the log scale. The median speedup is **158.77** times.

H.3 CIFAR-10 Classifiers

Relationship between Runtime and Regularization Tab. 4 and Tab. 5 show the runtime achieved by the proposed method at different ℓ_1 regularization on the CIFAR-10 classifiers.

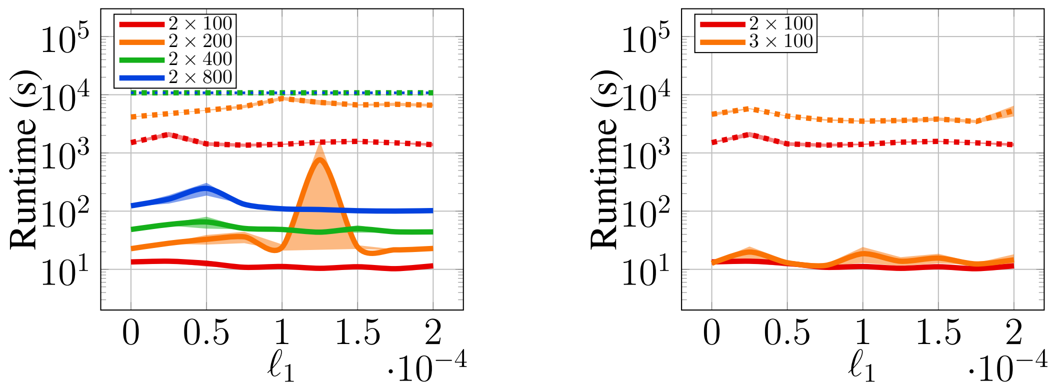
Table 4: **CIFAR10 Classifiers:** Compression results with fixed width and varying depth.

ARCH.	ℓ_1	ACCURACY (%)	COMPRESSION RUNTIME (S)	% REMOVED NEURONS	% REMOVED CONNECTIONS	TIMED OUT
2×100	0	54.3 ± 0.2	13.4 ± 0.6	0 ± 0	0 ± 0	0
2×100	0.000025	53.8 ± 0.9	14 ± 2	0 ± 0	0 ± 0	0
2×100	0.00005	53.6 ± 0.5	13 ± 3	31 ± 1	56 ± 2	0
2×100	0.000075	52.7 ± 0.6	10.9 ± 0.8	34 ± 2	61 ± 4	0
2×100	0.0001	52.3 ± 0.3	11 ± 2	36 ± 2	64 ± 2	0
2×100	0.000125	51.6 ± 0.5	10.4 ± 0.3	39 ± 3	66 ± 4	0
2×100	0.00015	51 ± 0.4	11 ± 2	40 ± 2	68 ± 3	0
2×100	0.000175	50.4 ± 0.4	10.3 ± 0.1	42 ± 3	69 ± 3	0
2×100	0.0002	50.1 ± 0.6	12 ± 2	45 ± 3	71 ± 3	0
2×100	0.000225	49.6 ± 0.4	11 ± 1	45 ± 2	72 ± 1	0
2×100	0.00025	48.5 ± 0.3	10.8 ± 0.7	46 ± 1	73 ± 2	0
2×100	0.000275	48 ± 0.4	10.3 ± 0.2	47 ± 3	75 ± 3	0
2×100	0.0003	47.8 ± 0.6	10.7 ± 0.6	51 ± 2	78 ± 2	0
2×100	0.000325	47.2 ± 0.2	10.4 ± 0.2	51 ± 3	77 ± 2	0
2×100	0.00035	47.2 ± 0.3	10.5 ± 0.5	53 ± 3	79 ± 3	0
2×100	0.000375	46.8 ± 0.4	10.7 ± 0.5	54 ± 3	80 ± 2	0
2×100	0.0004	46.3 ± 0.3	10.9 ± 0.4	56 ± 2	81 ± 2	0
3×100	0	53.7 ± 0.7	13 ± 1	0 ± 0	0 ± 0	0
3×100	0.000025	54.5 ± 0.4	20 ± 10	0 ± 0	0 ± 0	0
3×100	0.00005	53.8 ± 0.4	13 ± 2	22.3 ± 0.8	32 ± 1	0
3×100	0.000075	53.3 ± 0.6	11.6 ± 0.9	23 ± 2	34 ± 4	0
3×100	0.0001	53.2 ± 0.6	20 ± 10	25 ± 2	36 ± 3	0
3×100	0.000125	52.5 ± 0.6	14 ± 5	26 ± 2	38 ± 3	0
3×100	0.00015	51.98 ± 0.05	16 ± 6	29 ± 1	43 ± 1	0
3×100	0.000175	50.8 ± 0.6	12 ± 1	32 ± 2	47 ± 2	0
3×100	0.0002	50.3 ± 0.4	15 ± 7	35 ± 2	52 ± 3	0
4×100	0	53.6 ± 0.6	20 ± 10	0 ± 0	0 ± 0	0
4×100	0.000025	53.9 ± 0.6	20 ± 20	0 ± 0	0 ± 0	0
4×100	0.00005	53.9 ± 0.2	17 ± 8	15.9 ± 0.6	20.5 ± 0.8	0
4×100	0.000075	53.7 ± 0.3	13 ± 1	17 ± 1	22 ± 1	0
4×100	0.0001	52.7 ± 0.3	60 ± 90	19.3 ± 1	25 ± 1	0
4×100	0.000125	52.4 ± 0.6	15 ± 5	21 ± 2	29 ± 2	0
4×100	0.00015	51.6 ± 0.2	600 ± 800	25 ± 1	34 ± 2	0
4×100	0.000175	50.7 ± 0.3	700 ± 800	28.5 ± 0.9	40 ± 1	1
4×100	0.0002	50.3 ± 0.4	400 ± 400	33.7 ± 0.9	49 ± 1	0
5×100	0	53 ± 0.5	14.4 ± 0.4	0 ± 0	0 ± 0	0
5×100	0.000025	53.3 ± 0.8	18 ± 5	0 ± 0	0 ± 0	0
5×100	0.00005	54 ± 0.1	30 ± 20	12.9 ± 0.6	15.7 ± 0.7	0
5×100	0.000075	53.5 ± 0.4	100 ± 200	14 ± 0.5	17.1 ± 0.6	0
5×100	0.0001	53.3 ± 0.3	11.8 ± 0.4	16 ± 1	20 ± 1	2
5×100	0.000125	51.9 ± 0.4	3000 ± 4000	14 ± 8	20 ± 10	2
5×100	0.00015	51.4	1000	20	27	4
5×100	0.000175	51.3 ± 0.4	2000 ± 2000	27.4 ± 0.8	39 ± 1	3
5×100	0.0002	50.2 ± 0.1	3000 ± 2000	31 ± 2	45 ± 3	1

Runtime Comparison with SoTA Fig. 7 shows the comparison of runtime of the proposed method and the baseline with the strength of ℓ_1 regularization on the CIFAR-10 classifiers. We observe that the new method presents a median gain of **183.04** times in performance.

Table 5: **CIFAR10 Classifiers:** Compression results with fixed height and varying width.

Architecture	ℓ_1	Accuracy (%)	Compression Runtime (s)	% Removed Neurons	% Removed Connections	Timed Out
2×100	0	54.3 ± 0.2	13.4 ± 0.6	0 ± 0	0 ± 0	0
2×100	0.000025	53.8 ± 0.9	14 ± 2	0 ± 0	0 ± 0	0
2×100	0.00005	53.6 ± 0.5	13 ± 3	31 ± 1	56 ± 2	0
2×100	0.000075	52.7 ± 0.6	10.9 ± 0.8	34 ± 2	61 ± 4	0
2×100	0.0001	52.3 ± 0.3	11 ± 2	36 ± 2	64 ± 2	0
2×100	0.000125	51.6 ± 0.5	10.4 ± 0.3	39 ± 3	66 ± 4	0
2×100	0.00015	51 ± 0.4	11 ± 2	40 ± 2	68 ± 3	0
2×100	0.000175	50.4 ± 0.4	10.3 ± 0.1	42 ± 3	69 ± 3	0
2×100	0.0002	50.1 ± 0.6	12 ± 2	45 ± 3	71 ± 3	0
2×100	0.000225	49.6 ± 0.4	11 ± 1	45 ± 2	72 ± 1	0
2×100	0.00025	48.5 ± 0.3	10.8 ± 0.7	46 ± 1	73 ± 2	0
2×100	0.000275	48 ± 0.4	10.3 ± 0.2	47 ± 3	75 ± 3	0
2×100	0.0003	47.8 ± 0.6	10.7 ± 0.6	51 ± 2	78 ± 2	0
2×100	0.000325	47.2 ± 0.2	10.4 ± 0.2	51 ± 3	77 ± 2	0
2×100	0.00035	47.2 ± 0.3	10.5 ± 0.5	53 ± 3	79 ± 3	0
2×100	0.000375	46.8 ± 0.4	10.7 ± 0.5	54 ± 3	80 ± 2	0
2×100	0.0004	46.3 ± 0.3	10.9 ± 0.4	56 ± 2	81 ± 2	0
2×200	0	56.8 ± 0.2	23 ± 2	0 ± 0	0 ± 0	0
2×200	0.000025	56.8 ± 0.6	28 ± 1	0 ± 0	0 ± 0	0
2×200	0.00005	56.3 ± 0.4	30 ± 10	28 ± 2	54 ± 3	0
2×200	0.000075	55.5 ± 0.3	40 ± 20	32 ± 2	61 ± 3	0
2×200	0.0001	54.3 ± 0.4	24 ± 6	37 ± 2	68 ± 3	0
2×200	0.000125	53.3 ± 0.3	1000 ± 2000	42 ± 1	72 ± 2	0
2×200	0.00015	51.9 ± 0.7	24 ± 4	45 ± 2	75 ± 2	0
2×200	0.000175	51.2 ± 0.4	21.6 ± 0.8	49 ± 2	78 ± 2	0
2×200	0.0002	50.4 ± 0.1	23 ± 3	52 ± 2	80 ± 1	0
2×400	0	58.7 ± 0.1	48 ± 2	0 ± 0	0 ± 0	0
2×400	0.000025	59.2 ± 0.4	55 ± 9	0 ± 0	0 ± 0	0
2×400	0.00005	58.2 ± 0.1	60 ± 30	28 ± 1	54 ± 2	0
2×400	0.000075	56.1 ± 0.2	51 ± 3	37 ± 1	68 ± 2	0
2×400	0.0001	55 ± 0.3	48 ± 4	45 ± 2	75 ± 2	0
2×400	0.000125	53.5 ± 0.2	45 ± 3	48.3 ± 0.8	77.5 ± 0.6	0
2×400	0.00015	51.9 ± 0.3	50 ± 10	52 ± 1	80 ± 2	0
2×400	0.000175	50.9 ± 0.5	43 ± 3	56 ± 2	83 ± 1	0
2×400	0.0002	50.3 ± 0.3	45 ± 3	58 ± 3	83 ± 2	0
2×800	0	60.3 ± 0.2	125 ± 7	0 ± 0	0 ± 0	0
2×800	0.000025	60.3 ± 0.2	190 ± 80	0 ± 0	0 ± 0	0
2×800	0.00005	58.3 ± 0.2	240 ± 90	23 ± 6	50 ± 10	0
2×800	0.000075	56.3 ± 0.3	150 ± 50	37 ± 9	60 ± 10	0
2×800	0.0001	54.6 ± 0.2	108 ± 9	40 ± 10	70 ± 10	0
2×800	0.000125	53.2 ± 0.5	130 ± 30	50 ± 2	76 ± 2	0
2×800	0.00015	51.8 ± 0.3	110 ± 10	52.5 ± 0.8	78.2 ± 0.7	0
2×800	0.000175	50.6 ± 0.4	99 ± 6	53 ± 1	78.7 ± 1	0
2×800	0.0002	50.3 ± 0.2	98 ± 6	54 ± 1	79 ± 1	0



(a) With width

(b) With depth

Figure 7: **CIFAR-10 Classifiers: Comparison of runtimes** for proposed method (solid) and baseline (dashed) with the strength of regularization to identify stable neurons: (a) with increasing width (b) with increasing depth. We report the average and the standard deviation of the runtime of models with five different initialization for each regularization. Note that the y-axis is in the log scale. The median speedup is **183.04** times.

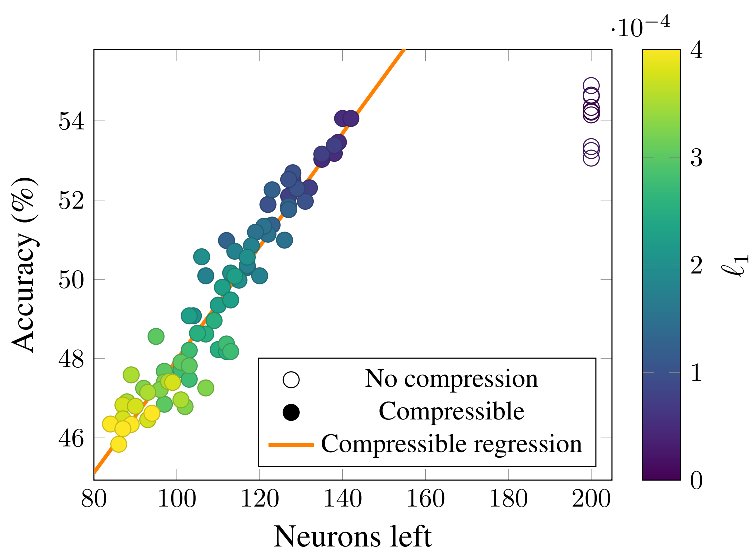


Figure 8: **Relationship between size of compressed neural network and accuracy on** 2×100 **CIFAR-10 classifiers.** The coefficient of determination (R^2) for linear regression obtained on the relationship between accuracy and neurons left for compressible networks is 91%.

H.4 CIFAR-100 Classifiers

Relationship between Runtime and Regularization Tab. 6 and Tab. 7 show the runtime achieved by the proposed method at different ℓ_1 regularization on the CIFAR-100 classifiers.

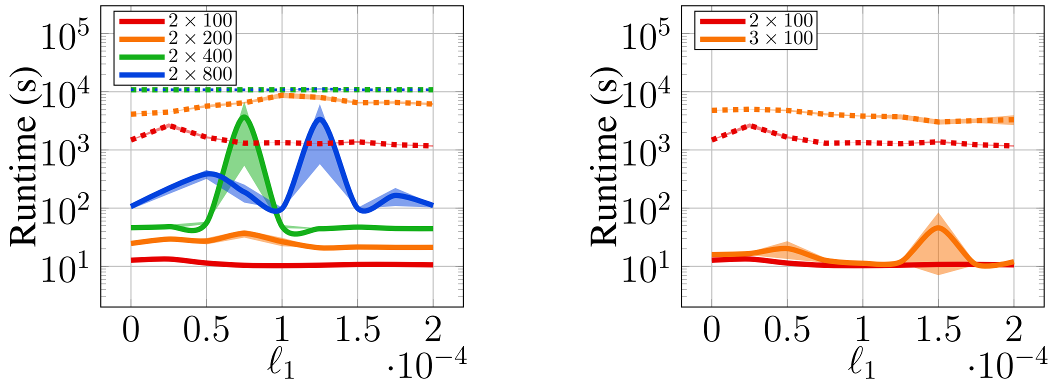
Table 6: **CIFAR100 Classifiers:** Compression results with fixed width and varying depth.

Arch.	ℓ_1	Accuracy (%)	Compression Runtime (s)	% Removed Neurons	% Removed Connections	Timed Out
2×100	0	25.2 ± 0.2	13 ± 1	0 ± 0	0 ± 0	0
2×100	0.000025	24.8 ± 0.4	13 ± 2	0 ± 0	0 ± 0	1
2×100	0.00005	24 ± 0.7	11.4 ± 0.4	36 ± 4	36 ± 4	1
2×100	0.000075	23.7	10.4	42	42	4
2×100	0.0001	23.4 ± 0.6	10.3 ± 0.1	42 ± 1	43 ± 1	1
2×100	0.000125	22 ± 2	10.453 ± 0.004	48 ± 1	48 ± 2	3
2×100	0.00015	22.4 ± 0.6	10.8 ± 1	48 ± 3	48 ± 3	1
2×100	0.000175	21.5 ± 0.5	10.8 ± 0.3	47.9 ± 0.2	48.7 ± 0.4	1
2×100	0.0002	21 ± 1	10.6 ± 0.3	51 ± 2	52 ± 2	0
2×100	0.000225	21.2 ± 0.4	11 ± 0.7	51 ± 2	52 ± 2	0
2×100	0.00025	21 ± 2	10.6 ± 0.5	50 ± 3	52 ± 4	0
2×100	0.000275	20.7 ± 0.8	10.4 ± 0.1	52 ± 2	54 ± 2	0
2×100	0.0003	19 ± 1	10.6 ± 0.2	53 ± 2	55 ± 2	0
2×100	0.000325	19 ± 1	10.7 ± 0.7	53 ± 4	55 ± 4	0
2×100	0.00035	19.2 ± 0.9	11 ± 1	53 ± 2	55 ± 1	0
2×100	0.000375	19.4 ± 0.5	10.5 ± 0.4	54 ± 2	56 ± 2	0
2×100	0.0004	19 ± 0.5	10.5 ± 0.3	53 ± 3	56 ± 3	0
3×100	0	24.9 ± 0.4	16 ± 3	0 ± 0	0 ± 0	0
3×100	0.000025	25.1 ± 0.4	17 ± 2	0 ± 0	0 ± 0	2
3×100	0.00005	25.4 ± 0.6	20 ± 10	22 ± 2	22 ± 2	2
3×100	0.000075	24 ± 1	13 ± 3	28 ± 2	28 ± 2	1
3×100	0.0001	24 ± 1	11.3 ± 0.4	30 ± 0.9	30.4 ± 1	1
3×100	0.000125	24 ± 1	12 ± 1	31 ± 1	32.4 ± 0.9	1
3×100	0.00015	23.1 ± 0.5	50 ± 80	34 ± 1	37 ± 1	0
3×100	0.000175	22 ± 1	10.7 ± 0.4	36 ± 2	38 ± 3	0
3×100	0.0002	22.4 ± 0.6	12 ± 1	39 ± 2	44 ± 3	0
4×100	0	24.7 ± 0.5	30 ± 20	0 ± 0	0 ± 0	0
4×100	0.000025	25 ± 0.7	16 ± 4	0 ± 0	0 ± 0	1
4×100	0.00005	24.8 ± 0.8	2000 ± 3000	18 ± 1	18 ± 1	1
4×100	0.000075	25.1 ± 0.5	12 ± 1	20 ± 1	20 ± 1	1
4×100	0.0001	24.8 ± 0.2	12 ± 2	22 ± 2	22 ± 2	2
4×100	0.000125	23.9 ± 0.4	11.8 ± 0.5	23.9 ± 0.4	25 ± 0.7	2
4×100	0.00015	23 ± 1	50 ± 70	28 ± 2	31 ± 3	1
4×100	0.000175	22 ± 2	50 ± 60	31 ± 3	36 ± 4	0
4×100	0.0002	22 ± 1	100 ± 200	34 ± 2	41 ± 2	0
5×100	0	24.2 ± 0.5	18 ± 4	0 ± 0	0 ± 0	0
5×100	0.000025	24.6 ± 0.4	100 ± 200	0 ± 0	0 ± 0	0
5×100	0.00005	25.4 ± 0.1	40 ± 30	12.9 ± 0.7	12.9 ± 0.7	3
5×100	0.000075	24.6 ± 0.2	14.1 ± 0.4	16.4 ± 0.3	16.6 ± 0.3	2
5×100	0.0001	24 ± 1	1000 ± 2000	18 ± 1	19 ± 2	1
5×100	0.000125	24.3 ± 0.2	200 ± 300	19 ± 1	20 ± 1	2
5×100	0.00015	23.6 ± 0.5	30 ± 20	22.2 ± 1	26 ± 2	0
5×100	0.000175	22 ± 1	1000 ± 1000	26.5 ± 0.5	32.4 ± 0.7	0
5×100	0.0002	22 ± 1	1000 ± 2000	31 ± 1	39 ± 1	1

Runtime Comparison with SoTA Fig. 9 shows the comparison of runtime of the proposed method and the baseline with the strength of ℓ_1 regularization on the CIFAR-100 classifiers. We observe that the new method presents a median gain of **137.29** times in performance.

Table 7: **CIFAR100 Classifiers:** Compression results with fixed height and varying width.

Architecture	ℓ_1	Accuracy (%)	Compression Runtime (s)	% Removed Neurons	% Removed Connections	Timed Out
2×100	0	25.2 ± 0.2	13 ± 1	0 ± 0	0 ± 0	0
2×100	0.000025	24.8 ± 0.4	13 ± 2	0 ± 0	0 ± 0	1
2×100	0.00005	24 ± 0.7	11.4 ± 0.4	36 ± 4	36 ± 4	1
2×100	0.000075	23.7	10.4	42	42	4
2×100	0.0001	23.4 ± 0.6	10.3 ± 0.1	42 ± 1	43 ± 1	1
2×100	0.000125	22 ± 2	10.453 ± 0.004	48 ± 1	48 ± 2	3
2×100	0.00015	22.4 ± 0.6	10.8 ± 1	48 ± 3	48 ± 3	1
2×100	0.000175	21.5 ± 0.5	10.8 ± 0.3	47.9 ± 0.2	48.7 ± 0.4	1
2×100	0.0002	21 ± 1	10.6 ± 0.3	51 ± 2	52 ± 2	0
2×100	0.000225	21.2 ± 0.4	11 ± 0.7	51 ± 2	52 ± 2	0
2×100	0.00025	21 ± 2	10.6 ± 0.5	50 ± 3	52 ± 4	0
2×100	0.000275	20.7 ± 0.8	10.4 ± 0.1	52 ± 2	54 ± 2	0
2×100	0.0003	19 ± 1	10.6 ± 0.2	53 ± 2	55 ± 2	0
2×100	0.000325	19 ± 1	10.7 ± 0.7	53 ± 4	55 ± 4	0
2×100	0.00035	19.2 ± 0.9	11 ± 1	53 ± 2	55 ± 1	0
2×100	0.000375	19.4 ± 0.5	10.5 ± 0.4	54 ± 2	56 ± 2	0
2×100	0.0004	19 ± 0.5	10.5 ± 0.3	53 ± 3	56 ± 3	0
2×200	0	28.2 ± 0.3	25 ± 3	0 ± 0	0 ± 0	0
2×200	0.000025	28.5	29.4	0	0	4
2×200	0.00005	28.1 ± 0.4	27 ± 7	31 ± 2	42 ± 3	0
2×200	0.000075	27.6 ± 0.3	40 ± 10	36 ± 1	48 ± 1	0
2×200	0.0001	26.9 ± 0.3	27 ± 9	40 ± 1	52 ± 1	0
2×200	0.000125	26.1 ± 0.3	20.8 ± 0.5	44 ± 2	57 ± 2	0
2×200	0.00015	25.7 ± 0.2	21 ± 1	46 ± 2	58 ± 2	0
2×200	0.000175	25 ± 0.3	21.1 ± 0.8	48 ± 2	60 ± 2	0
2×200	0.0002	24.2 ± 0.4	21.2 ± 0.6	49.1 ± 0.9	61.6 ± 0.9	1
2×400	0	30.2 ± 0.2	46.2 ± 0.8	0 ± 0	0 ± 0	1
2×400	0.000025	30.71 ± 0.04	51 ± 7	0 ± 0	0 ± 0	2
2×400	0.00005	30.2 ± 0.3	60 ± 10	26.5 ± 0.8	42 ± 1	1
2×400	0.000075	29.13 ± 0.09	49 ± 5	33 ± 2	51 ± 3	2
2×400	0.0001	28 ± 0.4	51 ± 7	38.3 ± 0.7	56.8 ± 0.9	1
2×400	0.000125	26.8 ± 0.4	44 ± 1	43 ± 1	62 ± 2	1
2×400	0.00015	25.9 ± 0.3	47 ± 3	45 ± 2	64 ± 2	1
2×400	0.000175	25 ± 0.2	44 ± 3	47 ± 1	66 ± 2	0
2×400	0.0002	24.2 ± 0.1	44 ± 2	48 ± 2	66 ± 2	0
2×800	0	31.32 ± 0.09	100 ± 20	0 ± 0	0 ± 0	2
2×800	0.00005	30.9 ± 0.3	300 ± 100	21.4 ± 0.8	38 ± 1	0
2×800	0.000075	29.4 ± 0.2	200 ± 100	32.5 ± 0.5	52.2 ± 0.8	0
2×800	0.0001	27.8 ± 0.3	97 ± 5	39.1 ± 0.7	60 ± 0.8	0
2×800	0.000125	26.7 ± 0.2	2000 ± 4000	41 ± 1	62 ± 1	0
2×800	0.00015	25.8 ± 0.2	98 ± 5	42 ± 1	64 ± 2	0
2×800	0.000175	24.6 ± 0.2	200 ± 100	44 ± 2	65 ± 2	0
2×800	0.0002	23.6 ± 0.5	110 ± 10	44.4 ± 1	66 ± 1	0



(a) With width

(b) With depth

Figure 9: **CIFAR-100 Classifiers: Comparison of runtimes** for proposed method (solid) and baseline (dashed) with the strength of regularization to identify stable neurons: (a) with increasing width (b) with increasing depth. We report the average and the standard deviation of the runtime of models with five different initialization for each regularization. Note that the y-axis is in the log scale. The median speedup is **137.29** times.

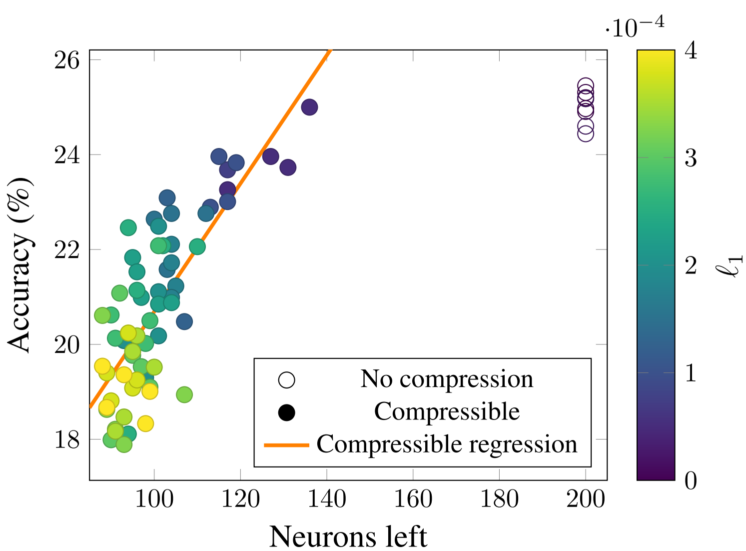


Figure 10: **Relationship between size of compressed neural network and accuracy on** 2×100 **CIFAR-100 classifiers.** The coefficient of determination (R^2) for linear regression obtained on the relationship between accuracy and neurons left for compressible networks is 61%.