Specification-Driven Neural Network Reduction for Scalable Formal Verification

Tobias Ladner
Technical University of Munich
Garching b. Munich, Germany
`tobias.ladner@tum.de`

Matthias Althoff
Technical University of Munich
Garching b. Munich, Germany
`althoff@tum.de`

Abstract

Formal verification of neural networks is essential before their deployment in safety-critical settings. However, existing methods for formally verifying neural networks are not yet scalable enough to handle practical problems that involve a large number of neurons. In this work, we propose a novel approach to address this challenge: A conservative neural network reduction approach that ensures that the verification of the reduced network implies the verification of the original network. Our approach constructs the reduction on-the-fly, while simultaneously verifying the original network and its specifications. The reduction merges all neurons of a nonlinear layer with similar outputs and is applicable to neural networks with any type of activation function such as ReLU, sigmoid, and tanh. Our evaluation shows that our approach can reduce a network to less than 5% of the number of neurons and thus to a similar degree the verification time is reduced.

1 Introduction

Neural networks achieve impressive results in a variety of fields such as autonomous cars [1]. However, the applicability of neural networks in safety-critical environments is still limited as small perturbations of the input can lead to unexpected outputs of the neural network [2]. Thus, the formal verification of neural networks gained importance in recent years [3], where approaches rigorously prove that the output of neural networks meets given specifications. These approaches are often based on satisfiability modulo theory solvers [4, 5], symbolic interval propagation [6, 7], or deploy reachability analysis [8, 9, 10, 11]. However, scalability is still a major issue for all of them [3].

While several approaches [12, 13] exist which reduce the size of the network by approximating the original network, to our best knowledge, there exist only a few reduction approaches with formal error bounds. An early approach [14] splits the neurons based on analytic properties and merges similar neurons afterward. This work is extended using interval neural networks [15, 16] and residual reasoning [17]. More closely related to our work is the approach in [18], which approximates the neural network output by merging neurons using clustering algorithms on a given dataset, however, $80 - 90\%$ of the neurons remain when formal error bounds are demanded. Most approaches only consider ReLU neurons; however, [15] also considers tanh neurons. A network reduction algorithm with formal error bounds for general neurons is still missing.

We propose a novel approach that reduces the neural network for given specifications. For example, consider an image as an input to a neural network. Neurons representing neighboring pixels often have similar values and thus can be merged during the verification process, which helps to reduce the high dimensionality of these neural networks. Such properties cannot be inferred when analyzing a neural network without considering a specific input. Our novel approach is orthogonal to many verification techniques, thus, many of them can be used as an underlying verification engine. We demonstrate our approach by deploying reachability analysis using zonotopes [9], the extension to other set-based verification tools is straightforward including Taylor models [19, 10] and polynomial zonotopes [11].

To summarize our main contributions, we present a novel, formally sound approach to reduce large neural networks by merging similar neurons for given specifications. The reduced network is constructed on-the-fly and the verification of the reduced network entails the verification of the original network. Our approach works on a variety of common activation functions, including ReLU, sigmoid, and tanh. The evaluation of high-dimensional datasets and benchmarks shows that the networks can be reduced to less than 5% of the original number of neurons using our novel approach. The overhead of constructing the reduced network is computationally cheap and thus the overall time for verifying the original network primarily depends on the number of remaining neurons in the reduced network.

The rest of this work is structured as follows. Sec. 2 introduces the notation and background for this work, then Sec. 3 presents our novel neuron-merging approach. We first show how similar neurons can be merged with formal error bounds, followed by the algorithm to construct the reduced network on-the-fly while verifying the original network. Finally, we evaluate our approach in Sec. 4 and conclude this work in Sec. 5.

2 Preliminaries

2.1 Notation

We denote vectors by lower-case letters, matrices by upper-case letters, and sets by calligraphic letters. The i-th element of a vector $b \in \mathbb{R}^n$ is written as $b_{(i)}$. Consequently, $b_{2(i)}$ is the i-th element of a vector b_2. The element in the i-th row and j-th column of a matrix $A \in \mathbb{R}^{n\times m}$ is written as $A_{(i,j)}$, the entire i-th row and j-th column are written as $A_{(i,\cdot)}$ and $A_{(\cdot,j)}$, respectively. The concatenation of two matrices A and B is denoted by $[A\ B]$. Given $n \in \mathbb{N}$, then $[n] = \{1, \dots, n\}$. Let $\mathcal{C} \subseteq [n]$, then $A_{(\mathcal{C},\cdot)}$ denotes all rows $i \in \mathcal{C}$. The cardinality of a discrete set $\mathcal{C}$ is denoted by $|\mathcal{C}|$. Let $\mathcal{S} \subset \mathbb{R}^n$ be a set of dimension n, then $\mathcal{S}_{(i)}$ is its projection on the i-th dimension. Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, then $f(\mathcal{S}) = \{f(x) \mid x \in \mathcal{S}\}$. An interval with bounds $a, b \in \mathbb{R}^n$ is denoted by $[a, b]$.

2.2 Neural Networks

In this work, we describe the reduction of feed-forward neural networks [20, Sec. 5.1]). We further note that our approach is also applicable to convolutional neural networks [20, Sec. 5.5.8], as convolutional layers and subsampling layers, e.g. average pooling layers, can be transformed into linear layers as defined subsequently.

Definition 1. *(Layers of Neural Networks [20, Sec. 5.1]) Let v_k denote the number of neurons in a layer k and $h_{k-1} \in \mathbb{R}^{v_{k-1}}$ the input. Further, let $W \in \mathbb{R}^{v_k \times v_{k-1}}, b \in \mathbb{R}^{v_k}$, and $\sigma_k(\cdot)$ be the respective continuous activation function (e.g. sigmoid and ReLU), which is applied element-wise. Then, the operation $L_k : \mathbb{R}^{v_{k-1}} \to \mathbb{R}^{v_k}$ on layer k is given by*

$$L_k(h_{k-1}) = \begin{cases} W_k h_{k-1} + b_k, & \textit{if layer } k \textit{ is linear}, \\ \sigma_k(h_{k-1}), & \textit{otherwise.} \end{cases} \tag{1}$$

Definition 2. *(Neural Networks [20, Sec. 5.1]) Given K alternating linear and nonlinear layers, v_0 input and v_K output neurons, and let $x \in \mathbb{R}^{v_0}$ be the input and $y \in \mathbb{R}^{v_K}$ be the output of a neural network, then, a neural network Φ with $y = \Phi(x)$ can be formulated as*

$$h_0 = x, \quad h_k = L_k(h_{k-1}), \quad y = h_K, \qquad k = 1 \dots K. \tag{2}$$

We call the last linear and the last nonlinear layer output layers, all other layers are hidden layers. If all hidden layers output the same number of neurons, we write 6×200 to refer to a network with 6 linear and 6 nonlinear hidden layers with 200 neurons each.

2.3 Set-based Computation

We use sets for the formal verification of neural networks. Let $\mathcal{X}$ be the input set of the neural network. Then, the exact output sets of each layer are denoted by

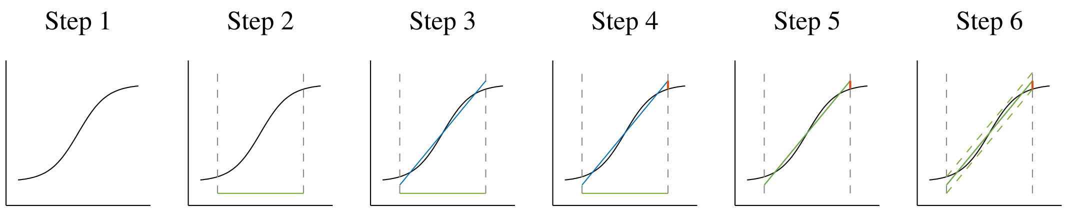

Figure 1: Main steps of enclosing a nonlinear layer. Step 1: Neuron-wise Sigmoid function. Step 2: Input bounds. Step 3: Approximation polynomial. Step 4: Approximation error. Step 5: Evaluate input on polynomial. Step 6: Add approximation error.

$$\mathcal{H}_0^* = \mathcal{X}, \quad \mathcal{H}_k^* = L_k(\mathcal{H}_{k-1}^*), \quad \mathcal{Y}^* = \mathcal{H}_K^*, \qquad k = 1 \dots K. \tag{3}$$

We use zonotopes as set representation to demonstrate our novel network reduction approach.

Definition 3. *(Zonotope [21, Def. 1]) Given a center vector $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times q}$, a zonotope is defined as*

$$\mathcal{Z} = \langle c, G \rangle_Z = \left\{ c + \sum_{j=1}^{q} \beta_j G_{(\cdot,j)} \,\middle|\, \beta_j \in [-1, 1] \right\}. \tag{4}$$

Further, we define two basic operations on zonotopes, which are essential for our approach. These operations can also be applied to many other set representations such as Taylor models and polynomial zonotopes.

Proposition 1. *(Interval Enclosure [22, Prop. 2.2]) Given a zonotope $\mathcal{Z} = \langle c, G \rangle_Z$, then the interval $[l, u] = \texttt{interval}(\mathcal{Z}) \supseteq \mathcal{Z}$ is given by*

$$\begin{matrix} l = c - \Delta g \\ u = c + \Delta g \end{matrix}\ , \quad \textit{with } \Delta g = \sum_{j=1}^{q} |G_{(\cdot,j)}|. \tag{5}$$

Proposition 2. *(Interval Addition [22, (2.1)]) Given a zonotope $\mathcal{Z} = \langle c, G \rangle_Z \subset \mathbb{R}^n$ and an interval $\mathcal{I} = [l, u] \subset \mathbb{R}^n$, then*

$$\mathcal{Z} \oplus \mathcal{I} = \langle c + c_\mathcal{I}, [G \ \texttt{diag}(u - c_\mathcal{I})] \rangle_Z \,, \tag{6}$$

where $c_\mathcal{I} = \frac{l+u}{2}$ and $\texttt{diag}$ creates a diagonal matrix.

2.4 Neural Network Verification

Finally, we briefly introduce the main steps to propagate a zonotope through a neural network. The propagation cannot be computed exactly in general, thus, the exact output of each layer needs to be enclosed.

Proposition 3. *(Image Enclosure [9, Sec. 3]) Let $\mathcal{H}_{k-1} \supseteq \mathcal{H}_{k-1}^*$ be an input set to layer k, then*

$$\mathcal{H}_k = \texttt{enclose}(L_k, \mathcal{H}_{k-1}) \supseteq \mathcal{H}_k^* \tag{7}$$

computes an over-approximative output set.

Linear layers can be computed exactly using zonotopes [21], however, nonlinear layers introduce over-approximations. We refer to [9, Sec. 3] for a detailed explanation, the main steps are as follows and are visualized in Fig. 1: For each nonlinear layer, we iterate over all neurons i in the current layer by projecting the input set $\mathcal{H}_{k-1}$ onto its i-th dimension (step 1) and determine the input bounds using Prop. 1 (step 2). We then find a linear approximation polynomial within the input bounds via regression [20, Sec. 3] (step 3). A key challenge is bounding the approximation error (step 4): For piece-wise linear activation functions, e.g. ReLU, we can compute the approximation error exactly using the extreme points of the difference between the approximation polynomial and each linear segment. For other activation functions, e.g. sigmoid, the approximation error can be determined by sampling evenly within the input bounds and bounding the approximation error between two points via global bounds of the derivative. Finally, we evaluate $\mathcal{H}_{k-1(i)}$ on the polynomial (step 5) and add the approximation error as an additional generator (step 6, Prop. 2).

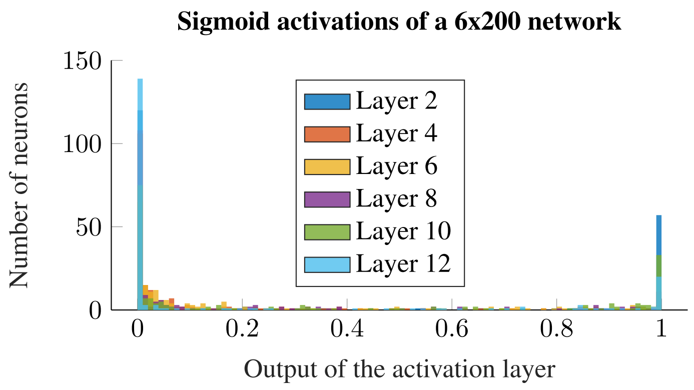

Figure 2: Sigmoid activations of a 6x200 neural network with an image input from the MNIST digit dataset. In this neural network, many neurons output values close to the saturation values 0 and 1.

2.5 Problem Statement

Given an input set $\mathcal{X} \subset \mathbb{R}^{v_0}$, a neural network Φ as specified in Def. 2, and an unsafe set $\mathcal{S} \subset \mathbb{R}^{v_K}$ in the output space of Φ, we want to find a reduced network $\widehat{\Phi}$ for which the verification entails the safety of the original network for $\mathcal{X}, \mathcal{S}$. Thus, it must hold

$$\widehat{\Phi}(\mathcal{X}) \cap \mathcal{S} = \emptyset \implies \Phi(\mathcal{X}) \cap \mathcal{S} = \emptyset. \tag{8}$$

3 Specification-Driven Neural Network Reduction

Our novel approach is based on the observation that many neurons in a layer k behave similarly for a specific input x, e.g. many sigmoid neurons are fully saturated and thus output a value near 1 as shown in Fig. 2. Neuron saturation [23], neural activation patterns [24], and over-parametrization [25] have been observed in the literature, however, to the best of our knowledge, it has not been considered during the verification of neural networks. Our main idea is to merge similar neurons such as these saturated neurons and provide the corresponding error bounds. Our novel approach is not restricted to the saturation values of an activation function but can merge neurons with any activation.

3.1 Neuron Merging

Subsequently, we explain how this observation can help to construct a much smaller network $\widehat{\Phi}$, where the verification of $\widehat{\Phi}$ entails the verification of the original network Φ. We denote the neurons which are merged using merge buckets:

Definition 4. *(Merge Buckets) Given output bounds $\mathcal{I}_k \supseteq \mathcal{H}_k^*$ of a nonlinear layer k with v_k neurons, an output $y \in \mathbb{R}$, and a tolerance $\delta \in \mathbb{R}$, then a merge bucket is defined as*

$$\mathcal{B}_{k,y,\delta} = \left\{ w \in [v_k] \;\middle|\; \mathcal{I}_{k(w)} \subseteq [y-\delta, y+\delta] \right\}. \tag{9}$$

Conceptually, we replace all neurons in $\mathcal{B}_{k,y,\delta}$ by a single neuron with constant output y and adjust the weight matrices of the linear layers $k-1, k+1$ such that the reduced network $\widehat{\Phi}$ approximates the behavior of the original network Φ. Finally, we add an approximation error to the output to obtain a sound over-approximation (Fig. 3). As the new neuron is constant, we can propagate it forward to the bias of the layer $k+1$ without inducing an over-approximation.

Proposition 4. *(Neuron Merging) Given a nonlinear hidden layer k of a network Φ, output bounds $\mathcal{I}_k \supseteq \mathcal{H}_k^*$, and a merge bucket $\mathcal{B}_{k,y,\delta}$, then we can construct a reduced network $\widehat{\Phi}$, where we remove the merged neurons by adjusting the linear layers $k-1$, $k-1$ such that*

$$\begin{aligned} \widehat{W}_{k-1} &= W_{k-1(\overline{\mathcal{B}}_{k,y,\delta},\cdot)}, & \widehat{b}_{k-1} &= b_{k-1(\overline{\mathcal{B}}_{k,y,\delta})}, \\ \widehat{W}_{k+1} &= W_{k+1(\cdot,\overline{\mathcal{B}}_{k,y,\delta})}, & \widehat{b}_{k+1} &= b_{k+1} + \underbrace{W_{k+1(\cdot,\mathcal{B}_{k,y,\delta})}\mathcal{I}_{k(\mathcal{B}_{k,y,\delta})}}_{\text{approximation error}}, \end{aligned} \tag{10}$$

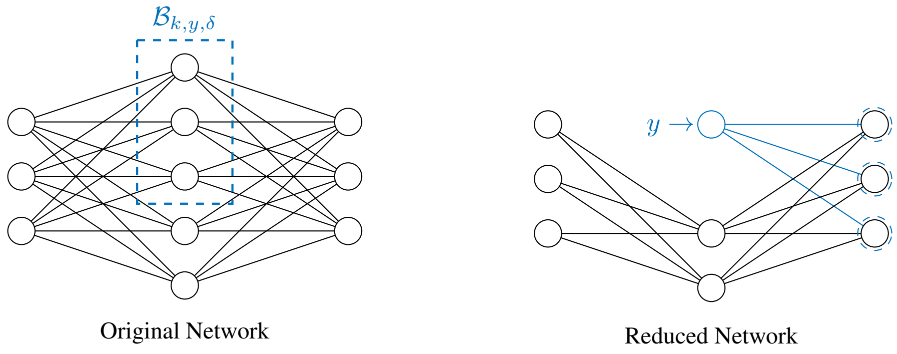

Figure 3: Neural network reduction example using a single merge bucket $\mathcal{B}_{k,y,\delta}$: All neurons within $\mathcal{B}_{k,y,\delta}$ get replaced by a single neuron with output y (in blue). An approximation error is added to the subsequent neurons.

where $\overline{\mathcal{B}}_{k,y,\delta} = [v_k] \backslash \mathcal{B}_{k,y,t}$ and $\widehat{b}_{k+1}$ includes the approximation error. We denote the layer operations of the reduced network $\widehat{\Phi}$ with $\widehat{L}_k$. The construction is sound.

Proof. Soundness. We show that the output $\widehat{\mathcal{H}}_{k+1}$ of layer $k+1$ of the reduced network $\widehat{\Phi}$ is an over-approximation of the exact set $\mathcal{H}^*_{k+1}$ as defined in (3). We drop the indices of $\mathcal{B}_{k,y,\delta}, \overline{\mathcal{B}}_{k,y,\delta}$ for conciseness:

$$\begin{aligned}
\mathcal{H}^*_{k+1} &\overset{(3)}{=} L_{k+1}\left(L_k\left(L_{k-1}(\mathcal{H}^*_{k-2})\right)\right) \overset{(1)}{=} L_{k+1}\left(L_k\left(W_{k-1}(\mathcal{H}^*_{k-2}) + b_{k-1}\right)\right) \\
&\overset{\text{(Def. 4)}}{=} L_{k+1}\left(L_k\left((W_{k-1(\mathcal{B},\cdot)}\mathcal{H}^*_{k-2} + b_{k-1(\mathcal{B})}) \times (W_{k-1(\overline{\mathcal{B}},\cdot)}\mathcal{H}^*_{k-2} + b_{k-1(\overline{\mathcal{B}})})\right)\right) \\
&\overset{(1),\,(10)}{=} L_{k+1}\left(L_k\left(\mathcal{H}^*_{k-1(\mathcal{B})}\right) \times \widehat{L}_k\left(\widehat{L}_{k-1}(\mathcal{H}^*_{k-2})\right)\right) \\
&\overset{\text{(Def. 4)}}{\subseteq} L_{k+1}\left(\mathcal{I}_{k(\mathcal{B})} \times \widehat{L}_k\left(\widehat{L}_{k-1}(\mathcal{H}^*_{k-2})\right)\right) \\
&\overset{(1)}{=} W_{k+1}\left(\mathcal{I}_{k(\mathcal{B})} \times \widehat{L}_k\left(\widehat{L}_{k-1}(\mathcal{H}^*_{k-2})\right)\right) + b_{k+1} \\
&\overset{\text{(Def. 4)}}{=} \left(W_{k+1(\cdot,\mathcal{B})}\mathcal{I}_{k(\mathcal{B})} \oplus W_{k+1(\cdot,\overline{\mathcal{B}})}\widehat{L}_k\left(\widehat{L}_{k-1}(\mathcal{H}^*_{k-2})\right)\right) + b_{k+1} \\
&= W_{k+1(\cdot,\overline{\mathcal{B}})}\widehat{L}_k\left(\widehat{L}_{k-1}(\mathcal{H}^*_{k-2})\right) \oplus \left(b_{k+1} + W_{k+1(\cdot,\mathcal{B})}\mathcal{I}_{k(\mathcal{B})}\right) \\
&\overset{(10)}{=} \widehat{L}_{k+1}\left(\widehat{L}_k\left(\widehat{L}_{k-1}(\mathcal{H}^*_{k-2})\right)\right) \overset{\text{(Prop. 3)}}{\subseteq} \widehat{\mathcal{H}}_{k+1},
\end{aligned} \tag{11}$$

where $\times$ denotes the Cartesian product and $\oplus$ denotes the Minkowski sum. □

In the extreme case of $\mathcal{B}_{k,y,\delta} = [v_k]$, thus all neurons of layer k output a value near y, the entire layer is simplified and our approach degenerates to pure interval arithmetic. However, the interval has very tight bounds per construction.

3.2 Bucket Creation

Prop. 4 can be naturally extended to multiple merge buckets, e.g.

$$\boldsymbol{\mathcal{B}}_k = \{\mathcal{B}_{k,0,\delta},\ \mathcal{B}_{k,1,\delta}\}, \tag{12}$$

where each neuron can only be part of one merge bucket $\mathcal{B} \in \boldsymbol{\mathcal{B}}$. A bucket $\mathcal{B}$ is only used if $|\mathcal{B}| > 1$. We define two different methods to create the merge buckets.

Static buckets. The merge buckets are determined by the asymptotic values of the respective activation function:

$$\boldsymbol{\mathcal{B}}_{\text{Sigmoid}} = \{\mathcal{B}_{\cdot,0,\delta},\ \mathcal{B}_{\cdot,1,\delta}\}, \quad \boldsymbol{\mathcal{B}}_{\text{Tanh}} = \{\mathcal{B}_{\cdot,-1,\delta},\ \mathcal{B}_{\cdot,1,\delta}\}, \quad \boldsymbol{\mathcal{B}}_{\text{ReLU}} = \{\mathcal{B}_{\cdot,0,\delta}\}. \tag{13}$$

Dynamic buckets. The merge buckets are dynamically created based on the output bounds $\mathcal{I}_k = [l_k, u_k] \subset \mathbb{R}^{v_k}$:

$$\boldsymbol{\mathcal{B}}_{k,\delta} = \left\{ \mathcal{B}_{k,c_{k(w)},\delta} \mid c_k = (l_k + u_k)/2,\ w \in [v_k] \right\}. \tag{14}$$

Bucket tolerance. The bucket tolerance δ is obtained by initially setting a large δ that allows for aggressive neuron merging and then decreasing δ adaptively if the specifications $\mathcal{S}$ are violated.

3.3 On-the-fly Neural Network Reduction

Note that we require output bounds $\mathcal{I}_k$ of the next nonlinear layer k to determine which neurons can be merged (Prop. 4). However, computing them requires the construction of high-dimensional zonotopes via the next linear layer $k-1$, propagating them through the nonlinear layer k — where we have to evaluate all neurons, which is what should be avoided. Thus, we deploy a look-ahead algorithm using interval arithmetic [26] to avoid these expensive computations and reduce the network on-the-fly. The algorithm is summarized in Alg. 1.

Algorithm 1 On-the-fly Neural Network Reduction

```
Require: Input $\mathcal{X}$, neural network layers $k \in [K]$, bucket tolerance $\delta$
 1: $\mathcal{H}_0 \leftarrow \mathcal{X}$
 2: $\widehat{L}_1 \leftarrow L_1$
 3: for $k = 2, 4, \dots, K$ do
 4:     if $k < K$ then                                                      ▷ Look ahead
 5:         $\mathcal{I}_{k-2} \leftarrow \texttt{interval}(H_{k-2})$                       ▷ Prop. 1
 6:         if $k > 2$ then $\mathcal{I}_{k-2} \leftarrow \mathcal{I}_{k-2} \cap \widehat{L}_{k-2}(\mathcal{I}_{k-3})$ end if    ▷ Tighten bounds
 7:         $\mathcal{I}_k \leftarrow L_k(\widehat{L}_{k-1}(\mathcal{I}_{k-2}))$
 8:         Create merge buckets $\boldsymbol{\mathcal{B}}_{k,\delta}$                            ▷ Sec. 3.2
 9:         $\widehat{L}_{k-1}, \widehat{L}_k, \widehat{L}_{k+1} \leftarrow$ Reduce network                   ▷ Prop. 4
10:     end if
11:                                                                          ▷ Verify
12:     $\mathcal{H}_{k-1} \leftarrow \texttt{enclose}(\widehat{L}_{k-1}, \mathcal{H}_{k-2})$                          ▷ Prop. 3
13:     $\mathcal{H}_k \leftarrow \texttt{enclose}(\widehat{L}_k, \mathcal{H}_{k-1})$
14: end for
15: $\mathcal{Y} \leftarrow \mathcal{H}_K$
16: return $\mathcal{Y}$
```

Instead of propagating the zonotope itself forward, we just propagate the interval bounds to the next nonlinear layer k (line 5-7). The overhead is computationally cheap and the bound computation is over-approximative. The bounds $\mathcal{I}_{k-2}$ can be intersected with $L_{k-2}(\mathcal{I}_{k-3})$ if $k > 2$ to get a tighter estimate (line 6), as $\mathcal{I}_{k-3}$ is computed anyway during the enclosure of the nonlinear layer $k-2$ (line 13; Fig. 1). After $\mathcal{I}_k$ is obtained, the merge buckets are determined (line 8) and the network is reduced by merging the respective neurons (line 9). Finally, we propagate the zonotope $\mathcal{H}_{k-2}$ on the reduced network. The addition of the interval bias of layer $k+1$ in (10) is computed using Prop. 2. Thus, we never construct a high-dimensional zonotope during the verification. Note that the number of input and output neurons remains unchanged.

Proposition 5. *(Reduced Network) Given a neural network Φ and an input set $\mathcal{X}$, then Alg. 1 constructs a reduced network $\widehat{\Phi}$, which satisfies the problem statement in Sec. 2.5.*

Proof. The algorithm is sound as each step is over-approximative. □

4 Evaluation

We evaluate our novel network reduction approach using several benchmarks and neural network variants from the VNN Competition [3]. For all datasets and networks, we sample 100 correctly classified images from the test set and average the results. All following figures show the mean remaining input neurons per layer $k \in [K]$ as well as the number of output neurons of the network

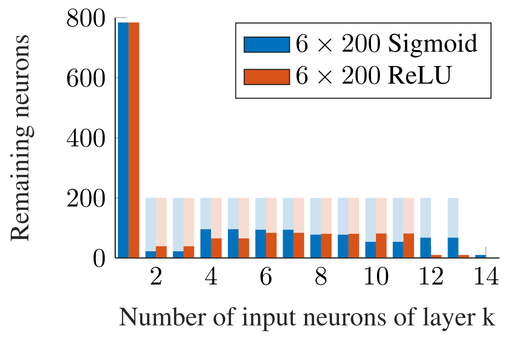

Figure 4: ERAN benchmark: Similar reduction rates for the both networks. The sigmoid network has an additional linear layer appended.

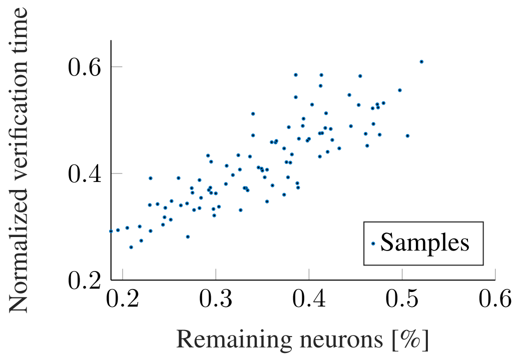

Figure 5: ERAN sigmoid network: The normalized verification time of the reduced network primarily depends on the reduction rate.

Table 1: ERAN benchmark: Change of verification rate (VR) and remaining neurons (RN) with varying perturbation radius r and bucket tolerance δ.

6 × 200 ReLU			
r	δ	RN [%]	VR [%]
0.0010	0.0001	38.59	100.00
0.0010	0.0010	37.83	100.00
0.0010	0.0050	36.70	99.00
0.0010	0.0100	37.88	96.00
0.0010	0.1000	58.25	4.00
0.0020	0.0001	46.44	100.00
0.0020	0.0010	46.63	100.00
0.0020	0.0050	46.64	100.00
0.0020	0.0100	47.40	96.00
0.0020	0.1000	55.01	12.00
0.0050	0.0001	57.77	100.00
0.0050	0.0010	55.95	100.00
0.0050	0.0050	57.35	100.00
0.0050	0.0100	56.60	99.00
0.0050	0.1000	59.65	27.00

6 × 200 Sigmoid			
r	δ	RN [%]	VR [%]
0.0010	0.0001	57.04	100.00
0.0010	0.0010	44.03	99.00
0.0010	0.0050	32.34	96.00
0.0010	0.0100	30.98	85.00
0.0010	0.1000	22.27	41.00
0.0020	0.0001	62.90	99.00
0.0020	0.0010	49.16	100.00
0.0020	0.0050	37.15	89.00
0.0020	0.0100	32.85	84.00
0.0020	0.1000	24.98	35.00
0.0050	0.0001	65.20	99.00
0.0050	0.0010	51.09	99.00
0.0050	0.0050	40.44	99.00
0.0050	0.0100	36.34	93.00
0.0050	0.1000	30.08	23.00

at $K+1$. The number of neurons of the original network is shown in the same color with reduced opacity and dynamic merge bucket creation (14) is used in all figures if not otherwise stated. We implement our approach in MATLAB and use CORA [27, 11] to verify the neural networks. All computations were performed on an Intel® Core™ Gen. 11 i7-11800H CPU @2.30GHz with 64GB memory.

4.1 MNIST Handwritten Digit Dataset

ERAN benchmark. We compare different network architectures using networks from the ERAN benchmark. Further network variants are taken from the ERAN website[1]. We compare different perturbation radii r and bucket tolerances δ using the sigmoid and ReLU networks from the original ERAN benchmark in Tab. 1. While it is expected that a more aggressive neuron merging (larger δ) results in fewer verified instances, we want to stress that using a very small δ can already result in many merged neurons. We observe similar results for the other benchmarks as well. All following figures use a perturbations radius $r = 0.001$ and a bucket tolerance $\delta = 0.005$. Fig. 4 shows the neuron reduction per layer for this combination. The number of remaining neurons in the reduced networks is very similar for both activation functions. The overhead of constructing the reduced

[1] ERAN website: `https://github.com/eth-sri/eran`

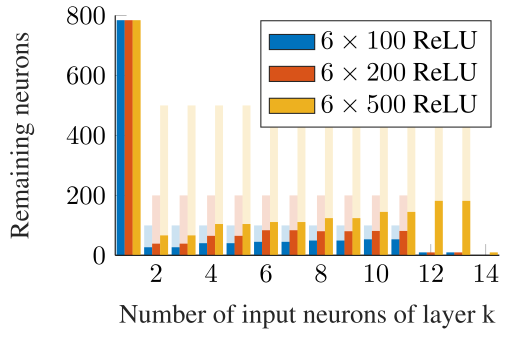

Figure 6: ERAN benchmark: The benefit of reducing neurons becomes more apparent with the size of the network.

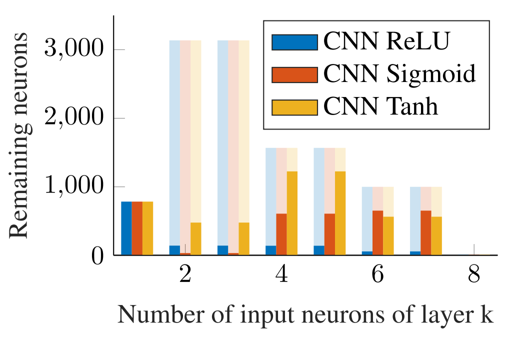

Figure 7: ERAN networks with convolutional layers: Huge reductions are possible as neighboring pixels are often very similar.

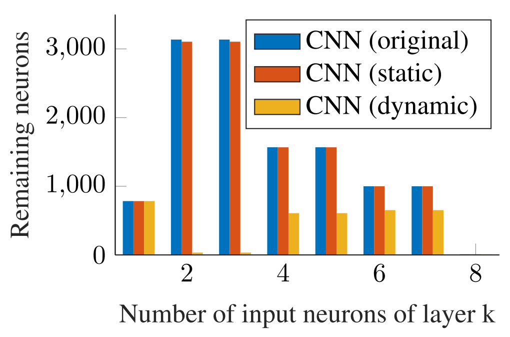

Figure 8: ERAN sigmoid network with convolutional layers: Dynamic merge buckets are required to merge similar pixels in images.

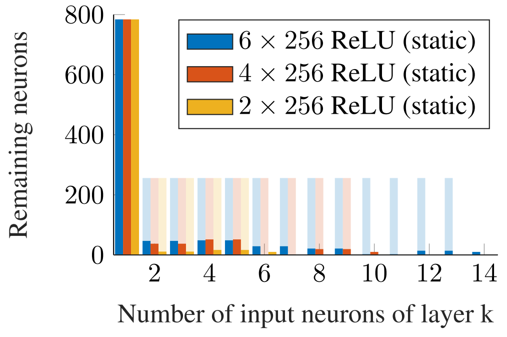

Figure 9: MNISTFC benchmark: Networks with large reduction with static merge buckets.

network is computationally cheap as shown in Fig. 5: We normalize the time to construct and verify the reduced network per image, where the verification of the original network takes $\sim 0.67s$. The verification time primarily depends on the remaining number of neurons. Our evaluation shows that the reduction on the ReLU network uses up to 5 dynamic merge buckets per layer and on the sigmoid network around 10 dynamic merge buckets per layer. Finally, Fig. 6 shows that the benefit of our novel reduction approach increases with the size of the network, as less neurons remain in the reduced network relative to the size of the original network.

ERAN convolutional neural networks. Convolutional layers can be transformed to linear layers as defined in this work (Def. 1). In convolutional neural networks, the neighboring pixel relation of images persists through convolutional layers. Thus, our approach is particularly applicable there as shown in Fig. 7, most notably in the sigmoid network where only 30 neurons of the ~ 3000 neurons remain in layer 2. As neurons representing neighboring pixels are not necessarily saturated, dynamic merge buckets are necessary as shown in Fig. 8. During the verification of convolutional neural networks, up to 200 dynamic merge buckets are created per layer.

MNISTFC benchmark. We want to stress that the reduction highly depends on the network. Although all networks presented here are trained with standard training, the output distribution of the nonlinear layers differs immensely. For example, not all networks with sigmoid activations have such extreme outputs as shown in Fig. 2. Fig. 9 shows another extreme case, where many ReLU neurons output zeros and thus can be removed. For this benchmark, we use the static merge buckets defined in (13) to only remove ReLU neurons with negative input. However, in general it is advisable to use dynamic merge buckets as previously shown in Fig. 8.

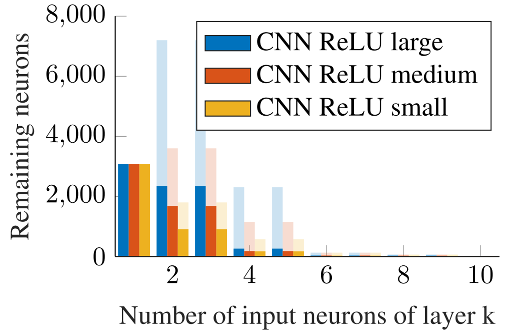

Figure 10: Marabou benchmark: Network reduction comparison on the CIFAR-10 dataset.

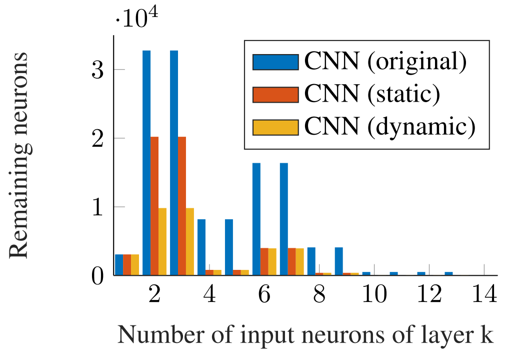

Figure 11: Cifar2020 benchmark: Our approach is also applicable to very large networks.

4.2 CIFAR-10 Colored Image Dataset

Finally, we show that our approach is also applicable to the CIFAR-10 colored image dataset. The size of networks trained on CIFAR-10 is typically much larger due to the complexity of the dataset. We note that the CIFAR-10 images are not normalized to $[0, 1]$.

Marabou benchmark. We show the network reduction on the Marabou benchmark in Fig. 10. The networks consist of two convolutional layers followed by three linear layers with ReLU activation. Our novel approach reduces these networks significantly, where more neurons are merged in larger networks.

Cifar2020 benchmark. The network consists of four convolutional layers with up to $32,768$ neurons per layer followed by three linear layers and ReLU activation. A dynamic merge bucket creation reduces the number of neurons to 25% while still verifying all images.

4.3 Discussion

We want to stress that our novel reduction method barely increases the over-approximation of the output $\mathcal{Y}$ by construction, as the over-approximation induced by the neuron merging is determined by the bucket tolerance δ. It is advisable to start with a large δ and automatically decrease δ until an image is verified. For example, consider the sigmoid network in Tab. 1: About 85% of the images can be verified with $\delta \geq 0.01$ in less than a third of the original verification time. While some networks have many activation neurons output near their asymptotic values, this is not the case for all networks. Thus, a dynamic creation of the merge buckets is important. Our approach is also applicable to convolutional neural networks as shown in the evaluation, which are usually especially high-dimensional. As neighboring pixels are often similar and remain similar through convolutional layers, our approach can reduce the dimensionality drastically for these networks.

5 Conclusion

We present a novel, conservative neural network reduction approach, where the verification of the reduced networks entails the verification of the original network. To our best knowledge, our approach is the first approach that works on a variety of activation functions and considers the specifications. The neural network reductions is computed on-the-fly while verifying the original network. Our approach merges neurons of nonlinear layers based on the output bounds of these neurons. These output bounds are computed by looking ahead to the next nonlinear layer using interval arithmetic, such that the more expensive verification algorithm only needs to be executed on the reduced network. Our novel approach is orthogonal to many verification tools and thus can be used in combination with them. We show the applicability of our approach on various benchmarks and network architectures. The overhead of computing the reduced network is not computationally expensive and the over-approximation of the output set barely increases. We believe that our work is a significant step toward more scalable neural network verification.

Acknowledgments and Disclosure of Funding

The authors gratefully acknowledge financial support from the project FAI funded by the German Research Foundation (DFG) under project number 286525601.

References

[1] Éloi Zablocki et al. “Explainability of deep vision-based autonomous driving systems: Review and challenges”. In: *International Journal of Computer Vision* 130.10 (2022), pp. 2425–2452.

[2] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *International Conference on Learning Representations*. 2015.

[3] Stanley Bak, Changliu Liu, and Taylor T. Johnson. “The second international verification of neural networks competition (VNN-COMP 2021): summary and results”. In: *arXiv preprint arXiv:2109.00498* (2021).

[4] Guy Katz et al. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *International Conference on Computer Aided Verification*. Springer. 2017, pp. 97–117.

[5] Guy Katz et al. “The marabou framework for verification and analysis of deep neural networks”. In: *International Conference on Computer Aided Verification*. Springer. 2019, pp. 443–452.

[6] Patrick Henriksen and Alessio Lomuscio. “Efficient neural network verification via adaptive refinement and adversarial search”. In: *ECAI 2020*. IOS Press, 2020, pp. 2513–2520.

[7] Gagandeep Singh et al. “An abstract domain for certifying neural networks”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–30.

[8] Timon Gehr et al. “Ai2: Safety and robustness certification of neural networks with abstract interpretation”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 3–18.

[9] Gagandeep Singh et al. “Fast and effective robustness certification”. In: *Advances in neural information processing systems* 31 (2018).

[10] Chao Huang et al. “POLAR: A polynomial arithmetic framework for verifying neural-betwork controlled systems”. In: *Automated Technology for Verification and Analysis*. Springer International Publishing, 2022, pp. 414–430.

[11] Niklas Kochdumper et al. “Open-and closed-loop neural network verification using polynomial zonotopes”. In: *arXiv preprint arXiv:2207.02715* (2022).

[12] Zhangheng Li et al. “Can pruning improve certified robustness of neural networks?” In: *arXiv preprint arXiv:2206.07311* (2022).

[13] Lei Deng et al. “Model compression and hardware acceleration for neural networks: A comprehensive survey”. In: *Proceedings of the IEEE* 108.4 (2020), pp. 485–532.

[14] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. “An abstraction-based framework for neural network verification”. In: *International Conference on Computer Aided Verification*. Springer. 2020, pp. 43–65.

[15] Fateh Boudardara et al. “Interval weight-based abstraction for neural network verification”. In: (International Conference on Computer Safety, Reliability, and Security). Springer. 2022, pp. 330–342.

[16] Pavithra Prabhakar and Zahra Rahimi Afzal. “Abstraction based output range analysis for neural networks”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[17] Yizhak Yisrael Elboher, Elazar Cohen, and Guy Katz. “Neural network verification using residual reasoning”. In: *arXiv e-prints* (2022), arXiv–2208.

[18] Pranav Ashok et al. “DeepAbstract: neural network abstraction for accelerating verification”. In: *International Symposium on Automated Technology for Verification and Analysis*. Springer. 2020, pp. 92–107.

[19] Sergiy Bogomolov et al. “JuliaReach: A toolbox for set-based reachability”. In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019.

[20] Christopher M. Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006.

[21] Antoine Girard. “Reachability of uncertain linear systems using zonotopes”. In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2005, pp. 291–305.

[22] Matthias Althoff. “Reachability analysis and its application to the safety assessment of autonomous cars”. PhD thesis. Technische Universität München, 2010.

[23] Anna Rakitianskaia and Andries Engelbrecht. “Measuring saturation in neural networks”. In: *2015 IEEE symposium series on computational intelligence*. IEEE. 2015, pp. 1423–1430.

[24] Alex Bäuerle, Daniel Jönsson, and Timo Ropinski. “Neural activation patterns (NAPs): Visual explainability of learned concepts”. In: *arXiv preprint arXiv:2206.10611* (2022).

[25] Behnam Neyshabur et al. “Towards understanding the role of over-parametrization in generalization of neural networks”. In: *arXiv preprint arXiv:1805.12076* (2018).

[26] Luc Jaulin et al. *Interval analysis*. Springer, 2001.

[27] Matthias Althoff. “An introduction to CORA 2015”. In: *Proc. of the workshop on applied verification for continuous and hybrid systems*. 2015, pp. 120–151.