Abstract—Formal verification of neural networks is essential before their deployment in safety-critical applications. However, existing methods for formally verifying neural networks are not yet scalable enough to handle practical problems involving a large number of neurons. We address this challenge by introducing a fully automatic and sound reduction of neural networks using reachability analysis. The soundness ensures that the verification of the reduced network entails the verification of the original network. To the best of our knowledge, we present the first sound reduction approach that is applicable to neural networks with any type of element-wise activation function, such as ReLU, sigmoid, and tanh. The network reduction is computed on the fly while simultaneously verifying the original network and its specifications. All parameters are automatically tuned to minimize the network size without compromising verifiability. We further show the applicability of our approach to convolutional neural networks by explicitly exploiting similar neighboring pixels. Our evaluation shows that our approach can reduce the number of neurons to a fraction of the original number of neurons with minor outer-approximation and thus reduce the verification time to a similar degree.

Index Terms—Formal verification, neural networks, set-based computation, neural network reduction, sound abstraction.

I. INTRODUCTION

Neural networks achieve impressive results in a variety of fields, including natural language processing [1], computer vision [2], and medical imaging [3]. In recent years, neural networks have been deployed in safety-critical environments, such as human-robot interaction [4] and autonomous driving [5]. As real-life applications are inherently exposed to noise, such as measurement inaccuracies and external disturbances, the deployment of neural networks in safety-critical environments is limited due to their sensitivity to adversarial attacks [6]: Even small perturbations of the input to a neural network, which are often barely noticeable to the human eye, can lead to unexpected outputs, e.g., a different predicted classification of an image or a controller returning an unsafe action. Thus, the formal verification of neural networks has gained importance in recent years [7]–[9], where approaches rigorously prove that the output of neural networks meets given specifications.

A. Related Work

Early approaches [10], [11] focus on complete algorithms to verify neural networks with ReLU activations, where either the specifications are formally proven or a counterexample is extracted. However, it has been shown that verifying a neural network with $n$ ReLU activations is NP-hard [11]. Thus, computing its exact output set for a given input set requires solving up to $2^n$ linear subproblems. Recent developments are made towards incomplete algorithms, where the neural networks are abstracted to enclose the exact behavior of the network. These approaches often formulate the formal verification of neural networks as an optimization problem [11]–[14] or use reachability analysis [15]–[23].

Optimization-based verifiers reason about neural networks by introducing relaxed, linear constraints for the activation functions and solving these relaxed problems using linear programming, satisfiability modulo theories (SMT) solvers [11]–[14], or symbolic interval propagation [24]–[26]. Branch-and-bound strategies [27] can be beneficial by splitting the problem at the neuron level [28], [29], e.g., by splitting ReLU neurons into their linear parts. In general, algorithms that split the problem lead to an exponential time complexity [11], so that current state-of-the-art tools [8] use advanced branch-and-bound strategies [30]–[32] to verify neural networks.

Verifiers using reachability analysis propagate sets through the neural network and verify given specifications using the computed outer-approximative output set. Simple representatives of this approach use pure interval arithmetic [33] or convex set representations such as zonotopes [15], [16]. As with optimization-based verifiers, splitting the set can improve the results [34], [35]. Non-convex set representations are used to tightly enclose the output due to the inherent nonlinearity of neural networks, including Taylor models [17]–[19], [36], star sets [20], [21], [37], and polynomial zonotopes [22], [23], [38]. However, the scalability to state-of-the-art networks remains a major challenge for optimization-based approaches and approaches based on reachability analysis [8].

One promising research direction for improving the scalability is sound neural network reduction [39]–[42], taking advantage of the typical over-parametrization of neural networks [43]. Sound neural network reduction reduces the number of neurons and provides formal bounds for the maximum error due to this reduction to reason about the original network. This research direction is closely related to neural network compression [44], [45], where the main goal is to reduce memory usage and computation time, e.g., for deployment in embedded systems [45]. Examples of compression techniques are quantization [46] and pruning [47]. However, the lack of formal error bounds prevents applying these techniques to the formal verification of neural networks.

To the best of our knowledge, there exist only a few network reduction approaches with formal error bounds: An early approach categorizes neurons based on analytic properties and merges neurons of the same category afterward [39].
This work is extended using interval neural networks [40], [42], [48], [49], where the weights of a neural network are replaced with intervals during the sound reduction. It is worth mentioning that the reduced network can be re-enlarged using residual reasoning [50]. For ReLU networks, it is also possible to merge neurons that are entirely in the nonpositive or nonnegative region, respectively, without inducing outer-approximations [51]. Network reductions can also be achieved by clustering similar neurons for inputs of a given dataset [41]; however, 80 – 90% of the neurons remain when formal error bounds are demanded. Most approaches only consider ReLU neurons, while [42], [49] also consider odd and monotone activation functions as tanh. We present a sound network reduction algorithm with formal error bounds for general element-wise activation functions.

B. Contributions

Our proposed approach reduces a neural network by merging similar neurons for given specifications. For example, consider a noisy image as an input set to a convolutional neural network. Neurons representing neighboring pixels often have similar values and thus can be merged during the verification process, which helps to reduce the size of the neural network. Such properties cannot be inferred when analyzing a neural network without considering a specific uncertain input. Our approach is orthogonal to many verification techniques, thus, they can be used as an underlying verification engine. We demonstrate our approach using reachability analysis with zonotopes [16], [52]. The extension to other set-based verification tools is straightforward, including Taylor models [18], [19], [36], star sets [20], [21], [37], and polynomial zonotopes [22], [23], [38]. The resulting reduced network can also be exported and verified using optimization-based verification tools. Our main contributions are summarized as follows:

- We present a novel, fully automatic approach to soundly 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 is applicable to all neural networks with element-wise activation functions, including ReLU, sigmoid, and tanh.
- To the best of our knowledge, we present the first neural network reduction approach that explicitly considers convolutional neural networks.
- Although our approach requires computing a new reduced network for different specifications, we show applications where the reduced network can be successfully reused.
- We will publish our approach with the next release of CORA [53].

The remainder of this work is structured as follows: Sec. II introduces the notation and background for this work. Then, we present our novel, fully automatic, and sound network reduction approach in Sec. III. In Sec. IV, we discuss applications of our approach, including the reduction of convolutional neural networks and how the reduced network can be reused for changed specifications. Finally, we evaluate our approach in Sec. V and draw conclusions in Sec. VI.

II. Preliminaries

A. Notation

We denote scalars and vectors by lower-case letters, matrices by upper-case letters, and sets by calligraphic letters. The $i$-th element of a vector $v \in \mathbb{R}^n$ is written as $v(i)$, and 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 $i$-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 $[A B]$. For $n \in \mathbb{N}$, we denote the identity matrix by $I_n$, and we use the notation $[n] = \{1, \ldots, n\}$. Let $C \subseteq [n]$, then $A(C, \cdot)$ extracts all rows $i \in C$ in lexicographic order. We denote the cardinality of a discrete set $C$ by $|C|$. Let $S \subseteq \mathbb{R}^n$ be a continuous set, then $S|i$ is its projection on the $i$-th dimension. The set-based evaluation of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is written as $f(S) = \{f(x) | x \in S \}$. Given two sets $S_1, S_2$, then the Minkowski sum is denoted by $S_1 \oplus S_2 = \{s_1 + s_2 | s_1 \in S_1, s_2 \in S_2 \}$. The Cartesian product is written as $S_1 \times S_2 = \{[s_1^T s_2^T]^T | s_1 \in S_1, s_2 \in S_2 \}$. An interval with bounds $l, u \in \mathbb{R}^n$, where $l \leq u$ holds element-wise, is denoted by $[l, u]$. We write $\mathbb{R}_+$ to refer to all positive real numbers.

B. Neural Networks

We first introduce feed-forward neural networks [54, Sec. 5.1] in their standard form and discuss the sound reduction of convolutional neural networks [54, Sec. 5.5.6] in Sec. IV-A.

Definition 1 (Layers of Neural Networks [54, Sec. 5.1]). Let $n_{k-1}, n_k$ denote the number of input and output neurons of a layer $k$. Further, let $W_k \in \mathbb{R}^{n_{k} \times n_{k-1}}, b_k \in \mathbb{R}^{n_k}$, and $\sigma_k(\cdot)$ be the respective continuous activation function (e.g., sigmoid and ReLU), which is applied element-wise. Then, the operation $L_k : \mathbb{R}^{n_{k-1}} \rightarrow \mathbb{R}^{n_k}$ for a given input $h_{k-1} \in \mathbb{R}^{n_{k-1}}$ is computed by

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

Definition 2 (Neural Networks [54, Sec. 5.1]). Given $\kappa$ alternating linear and nonlinear layers, $n_0$ input and $n_\kappa$ output neurons, let $x \in \mathbb{R}^{n_0}$ be the input and $y \in \mathbb{R}^{n_\kappa}$ be the output of a neural network, we can formulate a neural network $\Phi$ with $y = \Phi(x)$ as follows:

$$h_0 = x,$$

$$h_k = L_k(h_{k-1}), \quad k \in [\kappa],$$

$$y = h_{\kappa}.$$

The last linear and last nonlinear layers are called output layers, all other layers are called hidden layers. If all hidden layers output the same number of neurons, we write $6 \times 200$ to refer to a network with 6 linear and 6 nonlinear hidden layers with 200 neurons each.
C. Set-Based Computing

We use sets for the formal verification of neural networks. Let \( \mathcal{X} \subset \mathbb{R}^{n_0} \) be the input set of a neural network \( \Phi \). Then, the exact output set \( \mathcal{Y} = \Phi(\mathcal{X}) \) is computed by

\[
\mathcal{H}_0 = \mathcal{X}, \\
\mathcal{H}_k = L_k(\mathcal{H}_{k-1}), \quad k \in [\kappa], \\
\mathcal{Y} = \mathcal{H}_\kappa.
\]

(1)

Our reduction approach works for any set representation that can be enclosed by an interval and that are closed under the Minkowski addition of intervals. We use zonotopes as an example to demonstrate our approach:

Definition 3 (Zonotope [52, 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

\[
Z = (c, G)_Z = \left\{ c + \sum_{j=1}^{q} \beta_j G_{(\cdot,j)} \bigg| \beta_j \in [-1,1] \right\}.
\]

For zonotopes, the required operations are computed as follows:

Proposition 1 (Interval Enclosure [55, Prop. 2.2]). Given a zonotope \( Z = (c, G)_Z \), the enclosing interval \( [l, u] = \text{interval}(Z) \supseteq Z \) is

\[
l = c - \Delta g, \\
u = c + \Delta g, \quad \text{with} \quad \Delta g = \sum_{j=1}^{q} |G_{(\cdot,j)}|.
\]

Proposition 2 (Interval Addition [55, Eq. 2.1]). Given a zonotope \( Z = (c, G)_Z \subset \mathbb{R}^n \) and an interval \( I = [l, u] \subset \mathbb{R}^n \),

\[
Z \oplus I = (c + c_I, [G \ \text{diag}(u - c_I)])_Z,
\]

where \( c_I = \frac{l+u}{2} \) and \( \text{diag}(\cdot) \) returns a diagonal matrix.

D. Neural Network Verification

We briefly introduce the main steps to propagate a zonotope through a neural network. Since the propagation in (1) cannot be computed exactly in general, we enclose the output of each layer:

Proposition 3 (Image Enclosure [16, Sec. 3]). Let \( \mathcal{H}_{k-1} \supseteq \mathcal{H}_{k-1}^* \) be an input set to layer \( k \), then

\[
\mathcal{H}_k = \text{enclose}(L_k, \mathcal{H}_{k-1}) \supseteq \mathcal{H}_k^*
\]

computes an outer-approximative output set.

While zonotopes can be propagated through linear layers exactly [52], the propagation through nonlinear layers has to be outer-approximative to ensure soundness. The main steps to enclose the output of nonlinear layers are illustrated 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 determining the input bounds using Prop. 1 (Step 2). We then find an approximating linear function within the input bounds via regression [54, Sec. 3] (Step 3). A key challenge is bounding the approximation error (Step 4): For piecewise 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 an analytic solution for specific polynomials, or sampling evenly within the input bounds and bounding the approximation error between two points via global bounds of the derivative [22, Sec. 3]. Finally, we apply the linear transformation on \( \mathcal{H}_{k-1}^* \) to approximate the nonlinear layer (Step 5) and enclose the activation function using the approximation error (Step 6; Prop. 2). Thus, by propagating a given input set \( \mathcal{X} \) through all layers of a neural network and enclosing their output sets using Prop. 3, we can enclose the exact output set by \( \mathcal{Y} = \mathcal{H}_\kappa \supseteq \mathcal{Y}^* = \Phi(\mathcal{X}) \).

E. Problem Statement

Given an input set \( \mathcal{X} \subset \mathbb{R}^{n_0} \), a neural network \( \Phi \), and an unsafe set \( \mathcal{S} \subset \mathbb{R}^{n_{\kappa}} \), we want to formally construct a sound reduced network \( \tilde{\Phi} \), for which the verification entails the verification of the original network for the given \( \mathcal{X} \) and \( \mathcal{S} \):

\[
\tilde{\Phi}(\mathcal{X}) \cap \mathcal{S} = \emptyset \implies \Phi(\mathcal{X}) \cap \mathcal{S} = \emptyset.
\]

(2)

III. AUTOMATIC NEURAL NETWORK REDUCTION

Our sound neural network reduction is based on the observation that many neurons in a layer \( k \) behave similarly for a specific input \( x \in \mathbb{R}^{n_k} \), e.g., many sigmoid neurons are fully saturated and thus output a value near 1 as shown in Fig. 2.

Neuron saturation [56] and neural activation patterns [57] have been observed in the literature, however, to the best of our knowledge, they have not been exploited for the verification of neural networks. Our main idea is to merge these saturated neurons and provide the corresponding error bounds for an uncertain input \( \mathcal{X} \subset \mathbb{R}^{n_0} \) (Fig. 3). Please note that our approach is not restricted to the saturation values of an activation function.
A. Neuron Merging

Subsequently, we explain how similar neurons can help to construct a reduced network \( \hat{\Phi} \), where the verification of \( \hat{\Phi} \) entails the verification of the original network \( \Phi \). We gather the neurons with similar values using merge buckets (Fig. 3):

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

\[
B_{k,y,\delta} = \{ i \in [\kappa_k] \mid \mathcal{I}_{k(i)} \subseteq [y - \delta, y + \delta] \}.
\]

Conceptually, we replace all neurons in a merge bucket \( B_{k,y,\delta} \) by a single neuron \( w' \) with constant output \( y \) and adjust the weight matrices of the linear layers \( k-1 \) and \( k+1 \) such that the reduced network \( \hat{\Phi} \) approximates the behavior of the original network \( \Phi \). Finally, we add an approximation error to the output to obtain a sound outer-approximation (Fig. 3).

**Proposition 4** (Neuron Merging). Given a nonlinear hidden layer \( k \in [\kappa] \) of a network \( \Phi \), output bounds \( \mathcal{I}_k \supseteq \mathcal{H}^*_k \), a merge bucket \( B \), and the indices of the remaining neurons \( \overline{\mathcal{B}} = [\kappa_k] \setminus \mathcal{B} \), we can construct a sound reduced network \( \hat{\Phi} \), where we remove the merged neurons by adjusting the linear layers \( k-1 \), \( k+1 \), and \( \hat{\mathcal{B}}_{k+1} \) includes the approximation error:

\[
\begin{align*}
\hat{W}_{k-1} &= W_{k-1}(\overline{\mathcal{B}}), \\
\hat{W}_{k+1} &= W_{k+1}(\overline{\mathcal{B}}), \\
\hat{b}_{k-1} &= b_{k-1}(\overline{\mathcal{B}}), \\
\hat{b}_{k+1} &= b_{k+1} + \hat{W}_{k+1}(\overline{\mathcal{B}})\mathcal{I}(\overline{\mathcal{B}}).
\end{align*}
\]

We denote the layer operations of the reduced network \( \hat{\Phi} \) with \( \hat{L}_k \). The construction is sound.

**Proof.** Soundness. We show that the output \( \hat{\mathcal{H}}_{k+1} \) of layer \( k+1 \) of the reduced network \( \hat{\Phi} \) is an outer-approximation of the exact set \( \mathcal{H}^*_{k+1} \):

\[
\hat{\mathcal{H}}_{k+1} = L_{k+1} \left( L_k \left( W_{k-1}(\mathcal{H}^*_k) + b_{k-1} \right) \right) = L_{k+1} \left( L_k \left( W_{k-1}(\mathcal{H}^*_k) + b_{k-1} \right) \right).
\]

Without loss of generality, we relabel the neurons such that \( B := \{ B \} \) and partition the neurons of layer \( k \) accordingly:

\[
\hat{\mathcal{H}}_{k+1} = L_{k+1} \left( L_k \left( (W_{k-1}(B), \mathcal{H}^*_k) + b_{k-1}(B) \right) \right) \times (W_{k-1}(B), \mathcal{H}^*_k + b_{k-1}(B))
\]

(Def. 1) \( L_{k+1} \left( L_k \left( \mathcal{H}^*_k \right) \right) \times \hat{L}_k \left( \hat{L}_k \left( \mathcal{H}^*_k \right) \right).
\]

Then we enclose all merged neurons by the given interval bounds:

\[
\hat{\mathcal{H}}_{k+1} \subseteq L_{k+1} \left( \mathcal{I}_k + \hat{L}_k \left( \hat{L}_k \left( \mathcal{H}^*_k \right) \right) \right) = \hat{\mathcal{H}}_{k+1},
\]

and propagate them forward to the next nonlinear layer \( k+1 \). This operation implicitly propagates the new constant neuron forward to the bias of the layer \( k+1 \) as well without inducing additional outer-approximations. Thus, using the identity \( W(\hat{L}_1 \times \hat{L}_2) = W_{(1,2)} \hat{L}_1 \oplus W_{(1,2)} \hat{L}_2 \), we obtain:

\[
\hat{\mathcal{H}}_{k+1} \subseteq L_{k+1} \left( \mathcal{I}_k + \hat{L}_k \left( \hat{L}_k \left( \mathcal{H}^*_k \right) \right) \right) = \hat{\mathcal{H}}_{k+1},
\]

and finally, rearranging the terms and enclosing the output of all reduced layers using Prop. 3 obtains:

\[
\hat{\mathcal{H}}_{k+1} = W_{k+1}(\mathcal{B}) \hat{L}_k \left( \hat{L}_k \left( \mathcal{H}^*_k \right) \right) \oplus (b_{k+1} + W_{k+1}(\mathcal{B})\mathcal{I}_k) = \hat{\mathcal{H}}_{k+1} \subseteq \hat{\mathcal{H}}_{k+1},
\]

which shows that \( \hat{\mathcal{H}}_{k+1} \subseteq \mathcal{H}^*_k \subseteq \hat{\mathcal{H}}_{k+1} \).
We define two different methods to initialize merge buckets:

\[ \mathcal{B}_{k,\delta} = \{ \mathcal{B}_{k,1,\delta}, \mathcal{B}_{k,2,\delta}, \ldots \} . \] (3)

The merging with multiple disjoint merge buckets can be done in parallel as the required adaptations of the adjacent linear layers do not interfere with each other. The overall approximation error is then given by the Minkowski sum of the individual approximation errors (Prop. 4):

\[ \bigoplus_{\mathcal{B} \in \mathcal{B}} W_{k+1}(.,\mathcal{B}) \mathcal{I}_k(\mathcal{B}) = W_{k+1}(.,\bigcup_{\mathcal{B} \in \mathcal{B}} \mathcal{B}) \mathcal{I}_k(\bigcup_{\mathcal{B} \in \mathcal{B}} \mathcal{B}). \] (4)

We define two different methods to initialize merge buckets.

a) Static buckets: The merge buckets are determined by the asymptotic values of the respective activation function \( \sigma_k \) of a nonlinear layer \( k \):

\[ \mathcal{B}_{k,\delta} = \begin{cases} \{ \mathcal{B}_{k,0,\delta}, \mathcal{B}_{k,1,\delta} \} & \text{if } \sigma_k(x) = \text{sigmoid}(x), \\ \{ \mathcal{B}_{k,-1,\delta}, \mathcal{B}_{k,1,\delta} \} & \text{if } \sigma_k(x) = \text{tanh}(x), \\ \{ \mathcal{B}_{k,0,\delta} \} & \text{if } \sigma_k(x) = \text{ReLU}(x). \end{cases} \] (5)

For ReLU layers, setting \( \delta = 0 \) and using static merge buckets results in no approximation error similar to the approach in [51], as only neurons with entirely negative input for the given input set \( \mathcal{X} \) are removed.

b) Dynamic buckets: The merge buckets are dynamically initialized using the center of the bounds \( \mathcal{I}_k = [l_k, u_k] \subset \mathbb{R}^{n_k} \) of each neuron:

\[ \mathcal{B}_{k,\delta} = \{ \mathcal{B}_{k,c(i),\delta} \mid c = \frac{l_k + u_k}{2}, \ i \in |n_k| \} , \] (6)

where we ensure that the buckets are disjoint and are only used if they contain multiple neurons. Please note that the buckets could also be created using clustering algorithms similar to the approach in [41]; however, we choose the center of each neuron directly to obtain a linear computational overhead. The computational overhead of clustering algorithms might be negligible for other underlying verification engines than the zonotope approach considered in this work.

### C. Automatic Determination of Bucket Tolerances

The bucket tolerance \( \delta \in \mathbb{R}_+ \) influences how many neurons are merged, where a larger value results in more aggressive neuron merging and thus a larger outer-approximation. However, determining a good value for \( \delta \) is tedious as it is not immediately clear how much the network is reduced for any given value for \( \delta \). Thus, we automatically determine \( \delta \) given the desired remaining number of neurons in Alg. 1 using a binary search algorithm. We denote the ratio of remaining neurons compared to the original network with the reduction rate \( \rho \in [0,1] \). To verify given specifications, we initially choose a very small \( \rho \) and iteratively increase it if the reduction is too outer-approximative. This realizes us to verify many specifications using a heavily reduced network (Sec. V), and thus to a similar degree the verification time is reduced. Once \( \rho = 1 \) is reached, the original network is used and no reduction is applied.

#### Algorithm 1 Automatic Determination of Bucket Tolerance

**Require**: Bounds \( \mathcal{I}_k \), reduction rate \( \rho \)

1: \( \delta_{\min} \leftarrow 0 \), \( \delta_{\max} \leftarrow 0.01 \)
2: \( \delta \leftarrow 0 + \delta_{\max} \)
3: \( \delta_{\max} \leftarrow 10 \times \delta_{\max} \)
4: Initialize merge buckets \( \mathcal{B}_{k,\delta_{\max}} \) \( \triangleright \) Sec. III-B
5: \( \hat{n}_k \leftarrow n_k - \sum_{\mathcal{B} \in \mathcal{B}_{k,\delta_{\max}}} |\mathcal{B}| \) \( \triangleright \) Remaining neurons
6: while \( \hat{n}_k/n_k > \rho \)
7: \( \delta \leftarrow (\delta_{\min} + \delta_{\max})/2 \)
8: Initialize merge buckets \( \mathcal{B}_{k,\delta} \) \( \triangleright \) Sec. III-B
9: \( \hat{n}_k \leftarrow n_k - \sum_{\mathcal{B} \in \mathcal{B}_{k,\delta}} |\mathcal{B}| \) \( \triangleright \) Remaining neurons
10: if \( \hat{n}_k < \rho n_k \)
11: \( \delta_{\max} \leftarrow \delta \) \( \triangleright \) Too many neurons merged
12: else
13: \( \delta_{\min} \leftarrow \delta \) \( \triangleright \) Too few neurons merged
14: end if
15: \( \text{while } \hat{n}_k/n_k \neq \rho \)
16: return \( \mathcal{B}_{k,\delta} \)

#### D. On-the-fly Neural Network Reduction

We require output bounds \( \mathcal{I}_k \) of the next nonlinear layer \( k \) to merge neurons with similar values using Prop. 4. However, computing them requires the construction of high-dimensional zonotopes via the linear layer \( k-1 \) and the propagation of the zonotopes through the nonlinear layer \( k \), where we have to compute the image enclosure for all neurons (Prop. 3) – which is what should be avoided. Thus, we deploy a one-step look-ahead algorithm (Alg. 2) using interval arithmetic [58] to avoid these expensive computations and reduce the network on the fly. As the look-ahead is just a single step, the computed bounds are tight and do not contribute to the wrapping effect.

We summarize Alg. 2 subsequently: Instead of propagating the zonotope itself forward, we just propagate interval bounds of \( \mathcal{H}_{k-2} \) to the next nonlinear layer \( k \) (line 4-5). Although intervals are not closed under the linear map, the output bounds of the linear layer \( k-1 \) are tight and the propagation...
through the nonlinear layer \( k \) does not induce additional outer-approximations. This realizes a tight computation of the output bounds \( \mathcal{I}_k \) with negligible computational overhead. After \( \mathcal{I}_k \) is obtained, the merge buckets are determined (line 6) and the network is reduced by merging the respective neurons (line 7). Finally, we propagate the zonotope \( \mathcal{H}_{k-2} \) through the reduced layers. Thus, we never construct a high-dimensional zonotope during the verification. Note that the number of input and output neurons remains unchanged.

**Theorem 1** (Sound Network Reduction). Given an input set \( \mathcal{X} \), a neural network \( \Phi \), and a reduction rate \( \rho \), Alg. 2 constructs a reduced network \( \hat{\Phi}_\rho \) satisfying the problem statement in Sec. II-E.

**Proof.** The algorithm is sound as each step is outer-approximative. \( \square \)

### IV. APPLICATIONS

In this section, we discuss applications of our novel neural network reduction approach and evaluate them in Sec. V.

#### A. Reduction of Convolutional Neural Networks

Convolutional neural networks are obtaining state-of-the-art results for image classification tasks [2]. However, neural networks for image classification are typically very large and thus particularly hard to verify. We show in this section that our novel neural network reduction approach can be naturally extended to convolutional networks. Let us start by introducing the main layer within a convolutional network:

**Definition 5** (Convolutional Layer [54, Sec. 5.5.6]). Given an input \( I \in \mathbb{R}^{c_I \times h_I \times w_I} \) and a kernel \( K \in \mathbb{R}^{c_K \times h_K \times w_K} \), a convolutional layer computes the output \( O \in \mathbb{R}^{c_O \times h_O \times w_O} \) for \( k \in [c_O], \ i \in [h_O], \ j \in [w_O] \) as follows:

\[
O(k,i,j) = \sum_{l=1}^{c_O} \sum_{m=1}^{h_O} \sum_{n=1}^{w_O} K(k,l,m,n) I(l,i+m,j+n),
\]

where \( h_O = h_I - (h_K - 1), \ w_O = w_I - (w_K - 1) \) and \( c_I, c_O \) are the number of input and output channels, respectively.

Convolutional layers can be viewed as linear layers as defined in Def. 1 with shared weights [54, Sec. 5.5.6]. Thus, the same operation as in Def. 5 can be computed by flattening the input image \( I \) into a vector:

\[
\vec{I} = [I_1 \ \ldots \ I_{c_I}]^T,
\]

with \( I_l = [I_{l,1,1} \ \ldots \ I_{l,h_I\cdot w_I}], \ l \in [c_I] \), and correctly populating each row of the sparse weight matrix \( W_K \in \mathbb{R}^{(c_O-h_O\cdot w_O) \times (c_K-h_K\cdot w_K)} \) with the kernel \( K \):

\[
W_K = \begin{bmatrix}
K(1,1,1,1) & K(1,1,1,2) & \ldots & 0 \\
0 & K(1,1,1,1) & \ddots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
0 & \ldots & \ldots & K(c_O,c_I,h_K,w_K)
\end{bmatrix}.
\]

An analogous conversion can be done for other typical layers within a convolutional network, such as subsampling and average pooling layers [54, Sec. 5.5.6].

One important property of convolutional networks is the preservation of neighborhood: As the same kernel is applied to the entire input, pixels of the output have similar values if the respective pixels in the input have similar values. Neighboring pixels having similar values are typical in the field of image classification because many images contain large areas or objects with a similar color. For example, the sky has similar shades of blue, and traffic signs typically have only one background color and one foreground color. It is thus important to use dynamic merge buckets (Sec. III-B) as these colors might not be at the saturation points of the activation function. To the best of our knowledge, our approach is the first to explicitly exploit this property of convolutional networks for sound neural network reduction.

Intuitively, an uncertain image is compressed into superpixels with formal error bounds. Let us demonstrate this by an example: CIFAR-10 images require \( 32 \times 32 \times 3 = 3072 \) input neurons to the network. However, many of these pixels have very similar values (Fig. 4). Thus, using our approach with dynamic merge buckets, we can compress an image as follows:

**Corollary 1** (Sound Compression). Given an uncertain image \( \mathcal{X} \subset \mathbb{R}^{3 \times 3 \times 1} \) and a reduction rate \( \rho \in [0,1] \), we can construct a neural network \( \hat{\Phi}_\rho \) that compresses this image with formal error bounds as follows: Let \( K \in \mathbb{R}^{3 \times 3 \times 1 \times 1} \) be a kernel of a convolutional layer, where

\[
K(k,l,1,1) = \begin{cases}
1 & \text{for } k = l, \quad k, l \in [3], \\
0 & \text{otherwise}
\end{cases}
\]

and \( \Phi \) be a neural network with two convolutional layers with kernel \( K \) and one ReLU activation. The reduced network \( \hat{\Phi}_\rho \) obtained by applying Thm. 1 using dynamic merge buckets, \( \mathcal{X} \), and \( \rho \), compresses the input \( \mathcal{X} \) with sound error bounds according to \( \rho \).

**Proof.** The original network \( \Phi \) computes the identity by construction. The image is compressed in the hidden layer of \( \hat{\Phi}_\rho \) (Thm. 1). The computed bounds ensure \( \mathcal{X} \subseteq \Phi_\rho(\mathcal{X}) \). \( \square \)

In the truck example in Fig. 4, for a perturbation radius \( \epsilon = 0.01 \) and a reduction rate \( \rho = 0 \), all 3072 neurons of the hidden layer of the compression network \( \hat{\Phi}_\rho \) are dynamically merged using 21 merge buckets. Thus, the image is compressed into a 21-dimensional space in the hidden layer of \( \hat{\Phi}_\rho \) and then re-enlarged to 3072 neurons in the output layer with added approximation error. Please note that usually, the image is not re-enlarged and the compressed image is passed to the next layer, where we again merge similar neurons using our approach – we just do this here for illustration purposes (Fig. 4): Due to the three color channels, the original truck image has 983 unique colors, which get compressed into 178 unique colors with formal error bounds. Similarly, the original horse image has 891 unique colors, which get compressed into 142 unique colors, again using 21 dynamic merge buckets.

In larger convolutional networks, a similar reduction happens in each hidden layer; however, these are usually not as easy to grasp visually due to the increased number of channels in hidden layers. Note that we can prepend the layers of the
compression network defined in Cor. 1 to any network as a preprocessing step to reduce the input dimension.

The required steps for this preprocessing are provided in Alg. 3: We first construct the compression network as in Cor. 1. As we only compute the layers of the reduced network in line 2, we only require the input set represented as an interval \( \mathcal{X}_{\text{int}} \). Thus, we can construct a new low-dimensional input \( \mathcal{H}_{-2} \) represented by the used set representation according to the remaining neurons. The output set is computed in line 5 by propagating the set through all remaining layers and reducing them on the fly (Alg. 2).

**Algorithm 3 Sound Compression Preprocessing**

**Require:** Input \( \mathcal{X}_{\text{int}} \), neural network \( \Phi_{\text{org}} \), reduction rate \( \rho \)

1. \( \mathcal{L}_{-2}, \mathcal{L}_{-1}, \mathcal{L}_0 \leftarrow \text{Construct } \Phi_{\text{pre}} \quad \triangleright \text{ Cor. 1} \)
2. \( \hat{\mathcal{L}}_{-2}, \hat{\mathcal{L}}_{-1}, \hat{\mathcal{L}}_0 \leftarrow \text{Reduce } \Phi_{\text{pre}} \) using \( \mathcal{X}_{\text{int}}, \rho \quad \triangleright \text{ Thm. 1} \)
3. \( \mathcal{H}_{-2} \leftarrow \langle 0 \rangle \mathcal{Z} \oplus \hat{\mathcal{L}}_{-2}(\mathcal{X}_{\text{int}}) \quad \triangleright \text{ Prop. 2} \)
4. \( \Phi_{\text{org}}(\cdot) \leftarrow \Phi_{\text{org}}(\hat{\mathcal{L}}_0(\hat{\mathcal{L}}_{-1}(\cdot))) \quad \triangleright \text{ Prepend layers} \)
5. \( \mathcal{Y} \leftarrow \text{Execute Alg. 2 using } \mathcal{H}_{-2}, \Phi_{\text{org}}, \rho \quad \triangleright \text{ Thm. 1} \)
6. return \( \mathcal{Y} \)

Input sets are often given as an interval [7], and thus, we are not required to initialize the high-dimensional input set using a more complex set representation, i.e., a zonotope in our case. We only use the more complex set representation in the low-dimensional space and initialize it in line 3 with \( \mathcal{H}_{-2} \). For zonotopes, this results in fewer generators as we create a new generator for each dimension in the respective interval (Prop. 2). Due to the on-the-fly reduction, the more complex set representation is kept in a low-dimensional space as by the time it arrives at a given layer, this layer is already reduced (Alg. 2). This becomes increasingly beneficial with the complexity of the set representation used to verify the network. Note that Alg. 3 works on any neural network, but is especially beneficial for convolutional networks because neighboring pixels often have similar values.

**B. Reusing Reduced Networks**

In general, our approach requires the computation of a new reduced neural network for different input sets. In this section, we highlight several applications where the reduced networks can be reused nevertheless:

1) **Branch-and-bound:** Current state-of-the-art tools, e.g., all top-ranked tools of the last VNN competition [8], verify a neural network by applying different kinds of branch-and-bound algorithms in the verification. Branch-and-bound algorithms [27] partition the verification problem into multiple simpler subproblems, solve them individually, and aggregate the results to reason about the overall problem. For example, splitting ReLU neurons into their linear parts [30] makes each subproblem simpler. This approach was later extended to other nonlinear functions [32].

Our novel reduction approach is orthogonal to these branch-and-bound algorithms and can be combined with them. Splitting a set usually requires more sophisticated set representations than zonotopes, as the splitted sets can, in general, no longer be represented by a zonotope. We can reuse the reduced network on all subsets of the input set since the reduced network does not depend on using a specific set representation:

**Corollary 2 (Reusing Reduced Network on Subsets).** Given a neural network \( \Phi \), an input set \( \mathcal{X} \), and a reduction rate \( \rho \), then a reduced network \( \Phi_\rho \) according to Thm. 1 can be reused for all \( \mathcal{X} \subseteq \mathcal{X} \).

**Proof.** The proof follows from the construction of \( \Phi_\rho \). □

2) **Closed-loop verification:** In closed-loop scenarios, a neural network is used as a controller in a dynamic system which is updated every \( \Delta t \). While branch-and-bound strategies work well in open-loop verification, other techniques are more common in closed-loop scenarios [9]. The issue with branch-and-bound strategies is that each subset has to be propagated according to a differential equation until the next network evaluation, where each subset might again get splitted. Therefore, many techniques use more sophisticated set representation [18], [21], [22] and improve the abstraction by enclosing nonlinear functions with higher-order polynomials [23].
One frequent goal in closed-loop verification is to show the stability of a given dynamic system over a specified time horizon. For example, the QUAD benchmark in the last ARCH competition [9] requires showing the stability of a neural-network-controlled quadrotor at a given altitude (Fig. 5). We can infer from the simulations that the state of the system barely changes over the last second. Thus, we can slightly enlarge the current reachable set at $t = 4s$ and use it to reduce the size of the network. This reduced network can then be reused in subsequent evaluations if the reachable set stays within the set used to reduce the network controller (Cor. 2).

3) Export of reduced network: As the reduced network can be reused as described above, we provide an interface to export a reduced network for later usage, e.g., to verify the reduced network using another verification tool. Please note that a reduced network is of the form given in Def. 2, with the exception that the bias of a linear layer is an interval (Prop. 4). As biases can be seen as additional inputs to the network, most verifiers can verify networks of this form, including optimization-based verifiers.

V. Evaluation

We evaluate our novel neural network reduction approach using several neural network variants and benchmarks from the VNN competition [7]. For all image datasets, we sample 100 correctly classified images from the test set and average the results. The perturbation radius $\epsilon \in \mathbb{R}_+$ is always stated with respect to the normalized images $X \subset [0, 1]^m$. All following figures show the mean remaining input neurons per nonlinear layer $k \in [\kappa]$ as well as the number of input and output neurons of the network at 1 and $\kappa + 1$, respectively. We do not show the number of input neurons of linear layers, as a preceding nonlinear layer does not change the number of neurons. The number of neurons of the original network is shown in the same color with reduced opacity. Additionally, we show error bars indicating one standard deviation from the mean reduction per layer. If not otherwise stated, feed-forward neural networks are reduced using static merge buckets and convolutional neural networks using dynamic merge buckets (Sec. III-B).

We implemented our approach in MATLAB and use CORA [22], [53] to verify the neural networks. All computations were performed on an Intel® Core™ Gen. 11 i7-11800H CPU @2.30GHz with 64GB memory. We first show that our underlying assumption (Fig. 2) also holds in practice by choosing a fixed bucket tolerance $\delta$, followed by automatically determining $\delta$ to obtain the desired network reduction $\rho$, where $\rho$ is automatically increased if the image could not be verified.

A. Feed-Forward Neural Networks

1) MNISTFC Benchmark: In our first experiment, we used networks taken from the MNISTFC benchmark [7]. The benchmark uses images from the MNIST handwritten digit dataset. The images have a perturbation radius $\epsilon = 0.01$, and we use a bucket tolerance $\delta = 0.01$ for our evaluation. In Fig. 6, we show the reduction results on three neural networks with $6 \times 256$, $4 \times 256$, and $2 \times 256$ neurons. The reduced network retains, on average, only a small fraction of the neurons, ranging from 5% to 15% depending on the size of the network, where more neurons remain in earlier layers and only a few in later layers. While our network reduction induces outer-approximation, we were still able to verify all images using the reduced networks.

2) ERAN Benchmark: The authors of [41] apply their reduction approach on multiple network variants of the ERAN benchmark\(^1\). They first show that large informal network reductions using a small perturbation radius $\epsilon = 0.001$ are possible. However, once formal guarantees are demanded, 80 – 90% of the neurons remain [41, Fig. 2 & Tab. 2]. We obtained very similar reduction rates compared to their results; however, our approach provides sound error bounds: Fig. 7 shows the reduction results using a bucket tolerance $\delta = 0.005$ for the ReLU network of the ERAN benchmark and the two network variants with $6 \times 100$, $6 \times 200$, and $6 \times 500$ neurons, respectively. As we are able to reduce neural networks with any element-wise activation function using our approach, we can additionally reduce the network with sigmoid activations from the ERAN benchmark. Fig. 8 shows that we obtain similar reduction results using a bucket tolerance $\delta = 0.005$ for both networks.

\(^1\) Variants taken from the ERAN website: https://github.com/eth-sri/eran
tolerances $\delta$. The verification rate is the ratio of images that were verifiable with the reduced network compared to the original network. We iteratively reduce the bucket tolerance $\delta$ to measure how different merging strategies affect the reduction and verification: For the sigmoid network, a more aggressive merging strategy (large $\delta$) results in fewer remaining neurons but yields a smaller total number of verified images. However, even for $\delta = 0.1$ over half of the images can be verified, and already more than 95% for the next smallest bucket tolerance. This tradeoff is less apparent for the ReLU network as most of the merged neurons in a nonlinear layer have an entirely negative input, thus, are merged regardless of the chosen bucket tolerance. The remaining variation is due to the neurons that have an input near 0. Note that a too aggressive merging strategy might lead to fewer remaining neurons in earlier layers, which can result in larger outer-approximation in later layers and thus fewer neurons being merged in total.

With these results, we apply our fully automatic approach from Sec. III-C to automatically determine the bucket tolerance $\delta$ to verify a given image. As the construction of the reduced network is computationally cheap, the verification time is reduced to a similar degree as shown in Fig. 9: We vary the desired reduction rate $\rho$ and show the resulting average relative time to reduce and verify a network compared to verifying the original network directly. The average times to verify the original ERAN sigmoid network and the convolutional variant are 0.97s and 3.76s, respectively. The surrounding region shows one standard deviation. Thus, we can verify most images using the reduced network for small $\rho$ (Tab. I) in significantly less time (Fig. 9) and can iteratively increase $\rho$ where the verification is more challenging. For a challenging MNIST image with label 6, Fig. 10 shows the computed outer-approximative output bounds using the ERAN sigmoid network for different reduction rates $\rho$. The bounds quickly converge with increasing $\rho$, and the image can be verified with $\rho \geq 30\%$ in this example.

### B. Convolutional Neural Networks

We demonstrate the unique advantage of our reduction approach on convolutional neural networks by explicitly exploiting similar neighboring pixels. The subsequent convolutional networks from ERAN are again trained on the MNIST handwritten image dataset, and the networks from the Marabou and Cifar2020 benchmarks are trained on the CIFAR-10 colored image dataset [7]. For all convolutional networks, we use dynamic merge buckets unless stated otherwise, normalize the input image, and repeat each experiment over 100 correctly classified images. To show that our underlying assumption of many pixels having similar colors holds in practice, we again first show the reduction results on the ERAN networks for a fixed bucket tolerance $\delta = \epsilon$. The results using our fully
automatic reduction approach are then shown on the Marabou and Cifar2020 benchmarks.

1) ERAN: Fig. 11 shows the necessity for dynamic merge buckets to exploit similar neighboring pixels for a convolutional neural network with sigmoid activations. While barely any neurons are merged using static merge buckets, we obtain huge reductions using dynamic merge buckets, especially in the second layer, while still verifying the images. Further, we show a comparison of the reduction using networks with different activation functions in Fig. 12. Interestingly, large reductions can be achieved for all networks in the second layer. For the ReLU network, we can maintain a low number of neurons in later layers, too, while they increase again for the sigmoid and tanh networks for fixed $\delta$. An equal reduction in all layers can be obtained using the fully automatic approach.

2) Marabou: We show the network reduction using our fully automatic approach on the Marabou benchmark in Fig. 13. The networks consist of two convolutional layers followed by three linear layers with ReLU activation. We depict the reduction results of our network reduction in Fig. 14 for two different perturbation radii $\epsilon \in [0.01, 0.001]$. While the reduction results depend on the perturbation radius $\epsilon$ of the input set $X$, the difference becomes less apparent in later layers.

3) Cifar2020: Next, we consider the networks from the Cifar2020 benchmark. This network is an order of magnitude larger than the other convolutional networks and is thus particularly hard to verify. The network consists of four convolutional layers with up to 32,768 neurons per layer followed by three linear layers and ReLU activation. We depict the reduction results of our network reduction in Fig. 14 for two different perturbation radii $\epsilon \in [0.01, 0.001]$. While the reduction results depend on the perturbation radius $\epsilon$ of the input set $X$, the difference becomes less apparent in later layers.

4) Compression network: Fig. 15 shows how the sound compression preprocessing of the input image (Alg. 3) can further reduce the overall network size. The prepended layers shown at $-1$ and $0$ only have very few remaining neurons, where we only show the number of neurons corresponding to the dimension of the constructed zonotope. Thus, the average total computation time is reduced from 4.59s to 0.78s as the initial reduction (Alg. 3, line 2) is computationally cheap and the representation of the involved sets is much smaller, i.e., the zonotopes have fewer generators, compared to verifying the original network. Our evaluation shows that, on average, only 28 input neurons remain for the ERAN CNN with sigmoid activation compared to the $28 \cdot 28 = 784$ input neurons of the original image (Fig. 15).

C. Resuing a Reduced Network

Finally, we give two examples where the reduced network was reused despite its input set restriction (Cor. 2).

1) ACAS Xu benchmark: We demonstrate the applicability of our approach on non-image data using the ACAS Xu benchmark [7]. The benchmark consists of multiple networks and properties used to verify turn advisories to an aircraft to avoid collisions. The networks have $6 \times 50$ hidden layers with
network reduction is computed on the fly while verifying given specifications on the original network and merges neurons of nonlinear layers based on the output bounds of these neurons. The reduced network is computationally cheap to construct and does not induce large outer-approximations compared to the original network. All parameters of our approach are automatically tuned to minimize the network size without compromising verifiability and is orthogonal to many verification tools and thus can be combined with them. Further, our approach is the first to address the unique challenges of convolutional neural networks by explicitly exploiting similar neighboring pixels. Moreover, we show how our reduced network can be reused despite its restriction on the input set during branch-and-bound algorithms and closed-loop verification. Our evaluation shows the applicability of our approach on various benchmarks and network architectures, where the size of the networks is drastically reduced, which also decreases the total computation time.

**ACKNOWLEDGEMENTS**

The authors gratefully acknowledge financial support from the project FAI funded by the German Research Foundation (DFG) under project number 286525601. We also want to thank our colleagues Lukas Koller and Mark Wetzlinger from our research group for their revisions of the manuscript.

**REFERENCES**


