Information Sciences 665 (2024) 120367

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

# Maximum output discrepancy computation for convolutional neural network compression ☆

Zihao Mo, Weiming Xiang *

*School of Computer and Cyber Sciences, Augusta University, 1120 15th St, Augusta, 30912, GA, United States*

A R T I C L E  I N F O

*Keywords:*
Reachability analysis
Convolutional neural network
Discrepancy computation
Neural network compression

A B S T R A C T

Network compression methods minimize the number of network parameters and computation costs while maintaining desired network performance. However, the safety assurance of many compression methods is based on a large amount of experimental data, whereas unforeseen incidents beyond the experiment data may result in unsafe consequences. In this work, we developed a discrepancy computation method for two convolutional neural networks by giving a concrete value to characterize the maximum output difference between the two networks after compression. Using Imagestar-based reachability analysis, we propose a novel method to merge the two networks to compute the difference. We illustrate reachability computation for each layer in the merged network, such as the convolution, max pooling, fully connected, and ReLU layers. We apply our method to a numerical example to prove its correctness. Furthermore, we implement our developed methods on the VGG16 model with the Quantization Aware Training (QAT) compression method; the results show that our approach can efficiently compute the accurate maximum output discrepancy between the original neural network and the compressed neural network.

## 1. Introduction

As a tool for solving complex problems, neural networks have been demonstrated to be robust and effective. The neural networks play an essential role in many applications, including pattern recognition [20], [17], image processing [23], [16], computer vision [11], [34], control systems [13], [10], etc. To solve the problem with increasing complexity, the neural network model's scale and complexity are also increasing. However, an increase in model scale and complexity means increased resource consumption. For example, training a large deep neural network requires more computing power [27], higher memory, and more energy consumption [36]. To cope with the problem of high resource consumption of large neural networks, researchers focus on model compression. Pruning [41] and quantization [48] are two state-of-art compression methods. Pruning focuses on reducing neurons and lowering the scale/depth of the model, which helps reduce model complexity and save resource consumption. Quantization focuses on converting high-precision parameters to low-precision parameters, such as converting float type to integer type, to lower memory usage and reduce energy consumption.

☆ This work was supported by the National Science Foundation, under NSF CAREER Award no. 2143351, NSF CNS Award no. 2223035, and NSF IIS Award 2331938.

* Corresponding author.
*E-mail addresses:* zmo@augusta.edu (Z. Mo), wxiang@augusta.edu (W. Xiang).

https://doi.org/10.1016/j.ins.2024.120367
Received 14 August 2023; Received in revised form 22 December 2023; Accepted 21 February 2024

Available online 27 February 2024

A recent survey [1] has integrated and compared many mainstream neural network model compression methods to demonstrate the effect of different methods on the accuracy of the network. In another survey [46], four methods of compression and acceleration are summarized, but it points out some potential problems, such as a sharp drop in the accuracy of the compressed network. All these surveys indicate that compression of the model will lead to a more or less reduction in accuracy. For application goals, model compression methods need to keep the loss of accuracy as small as possible while reducing the complexity of the model. To demonstrate the high efficiency and low accuracy loss of their compression methods, those papers in the surveys demonstrate the practical application of their methods by training on some public databases and performing experimental results. On a theoretical level, demonstrating effectiveness through experimental results is accepted by the community. However, at the cyber-physical system level, training and simulation results on existing data are insufficient to demonstrate their effectiveness in practical applications, especially regarding safety. In [39], the authors address and explain the necessity of safety verification in CPS, and in [40], they provide a reachable-set-based estimation method to verify safety properties. How to represent the discrepancy for the compressed network and prove that the compressed network still retains the safety of the source network has become an essential problem in the community.

Quantitative analysis is an intuitive and convincing method to show the discrepancy between two networks. The authors of [30] bring up a quantitative verification method to verify DNN by providing a probability of a property being violated. In this paper [37], the authors successfully quantify the difference between the two networks by computing the reachability set of the discrepancy between the two networks. They verify the relation between the output reachable set and safety specifications to prove whether the compressed network safety is assured. In another study [47], the authors use a reachability analysis method for safety verification on neural feedback systems by computing the exact output reachable set of the systems in zonotopes, which share a similar idea to solving safety verification problems by providing quantitative results. Our method aims to provide quantitative analysis to formally characterize the performance of those compression methods by focusing on the discrepancy with the original network.

In this paper, we propose a formal method to quantitatively characterize the difference interval between the outputs of two convolutional neural networks generated from the same inputs. Based on the previous work for constructing a merged neural network with two different feedforward neural networks, we extend this method to the convolutional neural network and compute the guaranteed output error of the compression. Reachability analysis on the merged neural network can be performed to obtain the guaranteed range at each dimension of the output. The remainder of the paper is organized as follows: Related works are given in Section 2. The methodology is explained in Section 3. The main results of compression difference interval computation are presented in Section 4. An introduction to engineering applications is given in Section 5. The conclusion is presented in Section 6. Our source code for experiments is available online.[1]

## 2. Related work

### 2.1. Convolutional neural network

Convolutional neural networks (CNNs) are given the name with “convolutional” because this type of neural network includes the operation of convolution. Compared to conventional neural networks, which only accept vectors as their input and perform matrix computation in each neuron, convolutional neural networks accept a higher dimension matrix without flattening it and use a small filter to extract features contained in the matrix with a more complex structure. In CNNs, the convolution layer helps extract detailed features in the shallow layer and abstract features in the deep layer, with a pooling operation shrinking down the matrix size. Since the CNNs retain the spatial characteristics of the matrix during the learning process, they can better learn the abstract features in the input, which is the advantage of CNNs performing better than conventional neural networks in image classification, pattern recognition, and other problems. But, one of the challenges of CNNs is the interpretability of the learning process, which is usually viewed as a black box. In safety-critical problems, users need to verify the network outputs fall into the safety range to ensure the system’s stability, while the uninterpretable CNNs counterwork the verification, which provokes a trend to verify the robustness of a neural network.

### 2.2. Set-based verification for convolutional neural network

Neural network (NN) has been proven to be an efficient tool for solving complex and challenging problems in numerous domains. However, due to their widespread use, more and more research has proposed that NNs are vulnerable to adversarial attacks. From [3], a well-trained CNN can be fooled to make incorrect predictions by perturbing a subtle input noise. Before the network’s vulnerability can be overcome, there are reservations about applying it in safety-critical industries, such as autonomous driving, robotics, medical surgery, etc. To improve the robustness and safety of NN, formal verification methods of NN have recently become an important topic. Because feedforward neural networks (FNNs) have simple operation and intuitive structure, many verification methods for FNN have been proposed, including model-agnostic reachability analysis (DeepAgn) [44], star-set reachability [29,31], mixed monotonicity [21], reachable set estimation for memristive complex-valued neural networks (MCVNNs) [49], etc. However, for CNNs, only a handful of methods can verify their robustness and safety. The winner ($\alpha, \beta$-CROWN) [35], [45] of the 3rd International Verification of Neural Network Competition (VNN-COMP’22) uses a bound propagation based method to efficiently perform branch-and-bound

[1] The source code for experiments is publicly available: https://github.com/aicpslab/merged-cnn.

(BaB) based verification of neural networks and reduce the loosing of bounds compared to the linear programming (LP) solver. The second winner (MN-BaB) of VNN-COMP'2022 [2] brings up the idea of combining the convex relaxations method and BaB method to reduce subproblems and efficiently solve remaining verification problems. And the third winner (VeriNet) of VNN-COMP'2022 [6], [7] extends symbolic interval propagation to locate counter-example and uses an adaptive splitting strategy to refine nodes with the greatest impact on the output of the network. One of the popularly used methods, called the set-based method, [28] uses reachable sets to reason about the overall robustness of the network. It brings up a new set representation called ImageStar, which can represent an infinite family of images. Compared with optimization-based methods, set-based methods use reachable sets to represent all possible input with attack distortion and possible outputs of a CNN with that input set. It can prove the robustness of the network under image attack within arbitrary linear constraints. In our work, we use a set-based approach and reachable set representation ImageStar to perform verification of robustness between two CNNs. We combine the two networks and construct a merged network to perform reachable set computation and obtain the output difference between the two networks.

### 2.3. Guaranteed error estimation

Research on the compression of neural networks has been going on for decades, and many efficient methods have been proposed, such as quantization [48], pruning [41], knowledge distillation [9], and so on. However, recent studies [26] have found that networks trained with abundant data are often susceptible to subtle changes. The network's output will vary unexpectedly due to a slight change in a parameter, and it may even produce inaccurate predictions. In [19], the authors proposed a set-boundary reachability method to verify the safety properties of neural networks by checking whether all output falls into the giving guaranteed safe set. To represent all the possible output, this paper [38] introduces the output reachable set estimation and verification method and applies it to multilayer perception neural networks. To characterize the difference between the compressed neural network and the original network, a recent study [37] proposes a novel concept called approximate bisimulation relation. For the simulation relation, considering two neural networks, any output from one network can be generated by the same input from the other network and vice versa. Furthermore, it proposes a neural network merging algorithm for FNNs to calculate the approximate bisimulation error, measuring the distance between two networks. And [42] introduces a concept called guaranteed error estimation of feedforward neural networks, which intends to provide the worst-case approximation error of a trained neural network with respect to a compact input set essentially containing an infinite number of values. In our work, we expand the idea of merging neural networks. We can merge two CNNs to generate a new merged network and perform convolution operation, max pooling operation, linear operation, and ReLU activation with parameters from two CNNs. Our method characterizes the guaranteed error estimation of the outputs between two CNNs generated from the same inputs and indicates the error range.

## 3. Methodology

This work aims to formally compute the discrepancy between two convolutional neural networks in compression procedures. In this section, we introduce the reachable set representation method and discuss the advantages of using ImageStar for reachable set computation. Afterward, we detail our merging and reachability computations methods for each layer between the two CNNs.

### 3.1. Reachable set representation

For a neural network, given a set of inputs, the set of all possible outputs the network can reach is the reachable set of the network. Reachability analysis is efficient in the safety verification problem because the safety verification problem can be transformed into examining the relationship between network reachable sets and safety sets describing the specified safety constraint. The typical reachable sets can be represented as polyhedral in high-dimensional space. There exist multiple methods to represent and compute reachable sets, such as the zonotope method used in DeepZ [24], the polytope method used in DeepPoly [25], and the set-based method used in NNV [32]. DeepZ and DeepPoly have good performances on quickly reachable set computations for feedforward neural networks. But, for the CNN, their computation output is not tight enough compared to the NNV method. However, due to the complexity of CNN, the reachable set computation of the NNV method is time-consuming, taking more than an hour to give a result for the VGG models. As for 2D inputs, [28] demonstrates a better method, called ImageStar, for robustness verification of CNN. Our work uses ImageStar to represent and compute a reachable set for a merged network based on the fitness between ImageStar representation and convolutional neural network.

**Definition 1.** [28] ImageStar $\Theta$ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^{h\times w\times nc}$ is the anchor image, $V = \{v_1, v_2, \cdots, v_m\}$ is a set of $m$ images in $\mathbb{R}^{h\times w\times nc}$ called generator images, $P : \mathbb{R}^m \leftarrow \{\top, \bot\}$ is a predicate, and $h, w, nc$ are the height, width, and number of channels of the images, respectively. The generator images are arranged to form the ImageStar's $h \times w \times nc \times m$ basis array. The set of images represented by the ImageStar is:

$$\Theta = \{x | x = c + \sum_{i=1}^{m}(\alpha_i v_i), \textbf{ such that } P(\alpha_i, \cdots, \alpha_m) = \top\}.$$

In this work, we restrict the predicates to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for $p$ linear constraints, $C \in \mathbb{R}^{p\times m}$, $\alpha$ is the vector of $m$-variables, i.e., $\alpha = [\alpha_1, \cdots, \alpha_m]^T$, and $d \in \mathbb{R}^{p\times 1}$. An ImageStar is an empty set if and only if $P(\alpha)$ is empty.

### 3.2. Merged neural network

The goal of merging two neural networks is to perform a reachable analysis of the difference between the two neural networks. Our method uses ImageStar representation for reachable set computation. It gives an output $\mathbf{y}^{\{L+1\}} = \mathbf{y}_1^{\{L\}} - \mathbf{y}_2^{\{L\}}$ to indicate the difference between the two networks $\mathcal{N}_1, \mathcal{N}_2$. Below, we give assumptions for the two neural networks $\mathcal{N}_1$ and $\mathcal{N}_2$.

*Assumption 1:* The following assumptions hold for two neural networks $\mathcal{N}_1$ and $\mathcal{N}_2$:

(1) The number of inputs of two neural networks are the same, i.e., $n_1^{\{0\}} = n_2^{\{0\}}$;
(2) The number of outputs of two neural networks are the same, i.e., $n_1^{\{L\}} = n_2^{\{L\}}$;
(3) The number of layers of two neural networks is the same, i.e., $L_1 = L_2 = L$;
(4) For each layer $l$, two neural networks perform the same operation.

**Remark 1.** Generally, the compression neural network has fewer layers than the original network. To ensure the (3) and (4) in *Assumption 1*, we can expand the compression network with an extra layer that has the same number of inputs and outputs and the same operation. The expanded layers are forced to pass the information to subsequent layers without any changes, i.e., in [37], the author introduced a method to expand the fully connected layer when one network has fewer hidden layers than the other one. They added an extra fully connected layer with identity weights and zero biases to match the number of hidden layers between the two networks. And for the convolutional layer, we can add $1 \times 1$ filters to pass the same data to the subsequent layer. With expanded layers, the compression neural network has the same number of layers as the original network, and the two neural networks perform the same operation at each layer $l$.

Next, we start by demonstrating the architecture of merged CNN and computation for each layer in CNNs.

#### 3.2.1. Structure of merged CNN

Formally, the merged convolutional neural network is described by the following equation:

$$\begin{cases} \mathbf{y}^{\{l\}} = \phi^{\{l\}}(\mathbf{x}^{\{0\}}), & l = 1 \\ \mathbf{y}^{\{l\}} = \phi^{\{l\}}(\mathbf{y}^{\{l-1\}}), & l = 2, 3 \ldots, L \\ \mathbf{y}^{\{L+1\}} = \phi^{\{L+1\}}(\mathbf{y}^{\{L\}}) \end{cases}, \tag{1}$$

where $\mathbf{x}^{\{0\}}$ is the input vector and $\mathbf{y}^{\{l\}}$ is the output vector at layer $l$. $\phi^{\{l\}}$ is the operation of layer $l$ and $L$ is the total number of layer in CNN. For weight layers such as convolution layer and fully connected layer, layer operation $\phi^{\{l\}}$ is defined as $\phi^{\{l\}} = \langle \mathbf{W}_1^{\{l\}}, \mathbf{b}_1^{\{l\}}, \mathbf{W}_2^{\{l\}}, \mathbf{b}_2^{\{l\}} \rangle$, where $\mathbf{W}_1^{\{l\}}, \mathbf{b}_1^{\{l\}}$ are the weights and biases of network $\mathcal{N}_1$ at layer $l$, and $\mathbf{W}_2^{\{l\}}, \mathbf{b}_2^{\{l\}}$ are the weights and biases of network $\mathcal{N}_2$ at layer $l$. For non-weight layers such as the max pooling layer and the ReLU activation function, layer operation $\phi^{\{l\}}$ is defined with its related merged layer operation. The $L+1$ layer is defined as $\phi^{\{L+1\}} = \langle \mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}} \rangle$, where $\mathbf{I}_{n_{in}}$ is an identity matrix with size $n_{in} \times n_{in}$ in which $n_{in}$ is the dimension of input. The mapping relation of the merged neural network is defined as $\Phi(\cdot) = \phi^{\{L+1\}} \circ \phi^{\{L\}} \circ \cdots \circ \phi^{\{1\}}(\cdot)$ which includes convolution, max pooling, fully connected and other layer operation and activation function of the neural network, representing all structural information of the two neural networks, and an extra comparison layer. Thus, the output reachable set of the CNN is defined below

$$\mathcal{Y}^{\{L+1\}} = \left\{ \mathbf{y}^{\{L+1\}} \in \mathbb{R}^{n^{\{L+1\}}} \,|\, \mathbf{y}^{\{L+1\}} = \Phi(\mathbf{x}), \mathbf{x} \in \mathcal{X} \right\}. \tag{2}$$

The input set, internal state set, and output set of a merged CNN is a product of two ImageStars set $\Theta_1 = \langle c_1, V_1, P_1 \rangle$, $\Theta_2 = \langle c_2, V_2, P_2 \rangle$. The merged set is defined as $\Theta_m = \langle c_m, V_m, P_m \rangle$. The operation of merging is defined as $\Theta_m = \Theta_1 \uplus \Theta_2$, where the operation $\uplus$ is defined as $c_m = [c_1^T, c_2^T]^T$, $V_m = [V_1^T, V_2^T]^T$, and $P_m = P_1 \cap P_2$.

**Example 1.** Assume we have an ImageStar set $\Theta_1 = \langle c_1, V_1, P_1 \rangle$, where $c_1 = [0, 2]$, $V_1 = [0, 1]$, and $P_1 \equiv [1, -1]^T \alpha \leq [1, 1]^T$, and an ImageStar set $\Theta_2 = \langle c_2, V_2, P_2 \rangle$, where $c_2 = [1, 3]$, $V_2 = [-1, 0]$, and $P_2 \equiv [1, -1]^T \alpha \leq [1, 1]^T$. Then, we merge these two ImageStar sets to form a new set $\Theta_m = \Theta_1 \uplus \Theta_2 = \langle c_m, V_m, P_m \rangle$, where

$$c_m = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix}, V_m = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, P_m = P_1 \cap P_2 \implies P_m \equiv \begin{bmatrix} 1 \\ -1 \end{bmatrix} \alpha \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Thus, the merged ImageStar set can be represented as

$$\Theta_m = \Theta_1 \uplus \Theta_2 = \langle c_m, V_m, P_m \rangle = \{x | x = \begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix} + \alpha \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \textbf{ such that } P_m \equiv \begin{bmatrix} 1 \\ -1 \end{bmatrix} \alpha \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \}.$$

The input set $\mathcal{X} \in \mathbb{R}^{n^{\{0\}}}$ is defined as $\mathcal{X} = \mathcal{X}_1 \uplus \mathcal{X}_2$, where $n^{\{0\}}$ is the dimension of the input. As the difference between two networks is with respect to the same input, $\mathcal{X}_1$ is the same as $\mathcal{X}_2$. The input vector is denoted by $\mathbf{x} \in \mathcal{X}$. The output set of layer $l$ is defined as $\mathcal{Y}^{\{l\}} = \mathcal{Y}_1^{\{l\}} \uplus \mathcal{Y}_2^{\{l\}}, \mathcal{Y}^{\{l\}} \in \mathbb{R}^{n^{\{l\}}}$, where $n^{\{l\}}$ is the dimension of the output at layer $l$. The output vector of layer $l$ is denoted by $\mathbf{y}^{\{l\}} \in \mathcal{Y}^{\{l\}}$.

In our work, the merged convolutional neural network is also applied on the input reachable set $\mathcal{X}$ and gives the output reachable set $\mathcal{Y}$ as the discrepancy between two networks, which is described as follows

$$\begin{cases} \mathcal{Y}^{\{l\}} = \phi^{\{l\}}(\mathcal{X}^{\{0\}}), & l = 1 \\ \mathcal{Y}^{\{l\}} = \phi^{\{l\}}(\mathcal{Y}^{\{l-1\}}), & l = 2, 3, \ldots, L \\ \mathcal{Y}^{\{L+1\}} = \phi^{\{L+1\}}(\mathcal{Y}^{\{L\}}) \end{cases}. \tag{3}$$

*3.2.2. Merged convolutional layer*

A conventional two-dimensional convolutional layer is followed with parameters: the filters $\mathbf{W} \in \mathbb{R}^{h \times w \times n_c \times n_f}$, the bias $\mathbf{b} \in \mathbb{R}^{1\times 1\times n_f}$, the padding size $P$, the stride $S$, and the dilation factor $D$, where $h, w, n_c$ are the height, width, and the number of channels of the filters respectively, and $n_f$ is the number of filters. The convolution operation $\phi_c$ is represented below:

$$\mathbf{y} = \phi_c(\mathbf{x}) = \mathbf{W} * \mathbf{x} + \mathbf{b}, \tag{4}$$

where $*$ is the convolution operation and the padding $P$, stride $S$, and dilation factor $D$ affect the exact convolution operation. Let $\mathcal{N}_1, \mathcal{N}_2$ be the two CNNs and consider the convolution layer of the two CNNs is merged, the merged convolutional layer $\phi_c(\cdot)$ and its reachability computation are given in the following definition.

**Theorem 1.** *The merged convolutional layer $\phi_c^{\{l\}}$ is defined as $\phi_c^{\{l\}} = \langle \mathbf{W}_{1,c}^{\{l\}}, \mathbf{b}_{1,c}^{\{l\}}, \mathbf{W}_{2,c}^{\{l\}}, \mathbf{b}_{2,c}^{\{l\}} \rangle$, where $\mathbf{W}_{1,c}^{\{l\}}, \mathbf{b}_{1,c}^{\{l\}}$ are the filters and biases of network $\mathcal{N}_1$ at layer $l$, and $\mathbf{W}_{2,c}^{\{l\}}, \mathbf{b}_{2,c}^{\{l\}}$ are the filters and biases of network $\mathcal{N}_2$ at layer $l$. The input set of layer $l$ is given as $\mathcal{Y}^{\{l-1\}} = \langle c_m^{\{l-1\}}, V_m^{\{l-1\}}, P_m^{\{l-1\}} \rangle$ and vector $\mathbf{y}^{\{l-1\}} \in \mathcal{Y}^{\{l-1\}}$ is a linear combination of the anchor image and generator images $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$.*

*The output reachable set $\mathcal{Y}^{\{l\}}$ is*

$$\mathcal{Y}^{\{l\}} = \phi_c^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \langle \phi_{c-c}^{\{l\}}(c_m^{\{l-1\}}), \phi_{c-V}^{\{l\}}(V_m^{\{l-1\}}), \phi_{c-P}^{\{l\}}(P_m^{\{l-1\}}) \rangle = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_c^{\{l\}}(\mathbf{y}^{\{l-1\}})\}. \tag{5}$$

*$\phi_{c-c}^{\{l\}}$ for anchor image $c_m$ is defined as:*

$$\phi_{c-c}^{\{l\}}(c_m^{\{l-1\}}) = \begin{bmatrix} \phi_{1,c-c}^{\{l\}}(c_1^{\{l-1\}}) \\ \phi_{2,c-c}^{\{l\}}(c_2^{\{l-1\}}) \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{1,c}^{\{l\}} * c_1^{\{l-1\}} + \mathbf{b}_{1,c}^{\{l\}} \\ \mathbf{W}_{2,c}^{\{l\}} * c_2^{\{l-1\}} + \mathbf{b}_{2,c}^{\{l\}} \end{bmatrix}, \tag{6}$$

*where $\phi_{i,c-c}^{\{l\}} = \langle \mathbf{W}_{i,c}^{\{l\}}, \mathbf{b}_{i,c}^{\{l\}} \rangle$ is the convolution operation with related parameters from network $\mathcal{N}_i$.*

*$\phi_{c-V}^{\{l\}}$ for generator images $V_m$ is defined as:*

$$\phi_{c-V}^{\{l\}}(V_m^{\{l-1\}}) = \begin{bmatrix} \phi_{1,c-V}^{\{l\}}(V_1^{\{l-1\}}) \\ \phi_{2,c-V}^{\{l\}}(V_2^{\{l-1\}}) \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{1,c}^{\{l\}} * V_1^{\{l-1\}} \\ \mathbf{W}_{2,c}^{\{l\}} * V_2^{\{l-1\}} \end{bmatrix}, \tag{7}$$

*where $\phi_{i,c-V}^{\{l\}} = \langle \mathbf{W}_{i,c}^{\{l\}} \rangle$ is the convolution operation with related parameters from network $\mathcal{N}_i$.*

*$\phi_{c-P}^{\{l\}}$ for linear constraint $P_m$ is defined as:*

$$\phi_{c-P}^{\{l\}}(P_m^{\{l-1\}}) = P_m^{\{l-1\}}. \tag{8}$$

*Thus, the output reachable set $\mathcal{Y}^{\{l\}}$ can also be represented as:*

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_c^{\{l\}} \begin{bmatrix} \mathbf{y}_1^{\{l-1\}} \\ \mathbf{y}_2^{\{l-1\}} \end{bmatrix}\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{1,c}^{\{l\}}(\mathbf{y}_1^{\{l-1\}}) \\ \phi_{2,c}^{\{l\}}(\mathbf{y}_2^{\{l-1\}}) \end{bmatrix}\}. \tag{9}$$

**Proof.** The anchor image $c_m^{\{l\}} = Conv(c_m^{\{l-1\}})$ where $Conv(\cdot)$ is the convolution operation. Thus, the anchor image of the merged convolutional layer is performed as below:

$$\phi_{c-c}^{\{l\}}(c_m^{\{l-1\}}) = [\phi_{1,c-c}^{\{l\}}(c_1^{\{l-1\}})^T, \phi_{2,c-c}^{\{l\}}(c_2^{\{l-1\}})^T]^T = [(\mathbf{W}_{1,c}^{\{l\}} * c_1^{\{l-1\}} + \mathbf{b}_{1,c}^{\{l\}})^T, (\mathbf{W}_{2,c}^{\{l\}} * c_2^{\{l-1\}} + \mathbf{b}_{2,c}^{\{l\}})^T]^T = [(c_1^{\{l\}})^T, (c_2^{\{l\}})^T]^T = c_m^{\{l\}}.$$

As for the generator images $V_m^{\{l\}} = ConvZeroBias(V_m^{\{l-1\}})$ where $ConvZeroBias$ is the convolution operation with zero bias. Thus, the generator images of the merged convolutional layer are performed as below:

$$\phi_{c-V}^{\{l\}}(V_m^{\{l-1\}}) = [\phi_{1,c-V}^{\{l\}}(V_1^{\{l-1\}})^T, \phi_{2,c-V}^{\{l\}}(V_2^{\{l-1\}})^T]^T = [(\mathbf{W}_{1,c}^{\{l\}} * V_1^{\{l-1\}})^T, (\mathbf{W}_{2,c}^{\{l\}} * V_2^{\{l-1\}})^T]^T = [(V_1^{\{l\}})^T, (V_2^{\{l\}})^T]^T = V_m^{\{l\}}.$$

And linear constraints remain the same $P_m^{\{l\}} = P_m^{\{l-1\}}$ because the convolution operation performs local element-wise multiplication on a local matrix and bias addition and has no effect on the constraint. Based on these, we can represent the vector $\mathbf{y}^{\{l\}}$ with the result $\mathbf{y}_1^{\{l\}}$, $\mathbf{y}_2^{\{l\}}$ from each network $\mathcal{N}_1$, $\mathcal{N}_2$:

$$\mathbf{y}^{\{l\}} = \phi_{c-c}^{\{l\}}(c_m^{\{l-1\}}) + \alpha\phi_{c-V}^{\{l\}}(V_m^{\{l-1\}}) = \begin{bmatrix} \phi_{1,c-c}^{\{l\}}(c_1^{\{l-1\}}) \\ \phi_{2,c-c}^{\{l\}}(c_2^{\{l-1\}}) \end{bmatrix} + \begin{bmatrix} \alpha\phi_{1,c-V}^{\{l\}}(V_1^{\{l-1\}}) \\ \alpha\phi_{2,c-V}^{\{l\}}(V_2^{\{l-1\}}) \end{bmatrix} = \begin{bmatrix} \phi_{1,c}^{\{l\}}(\mathbf{y}_1^{\{l-1\}}) \\ \phi_{2,c}^{\{l\}}(\mathbf{y}_2^{\{l-1\}}) \end{bmatrix}. \tag{10}$$

Thus, for each vector $\mathbf{y}^{\{l-1\}}$ in the input set $\mathcal{Y}^{\{l-1\}}$, it has a corresponding output vector $\mathbf{y}^{\{l\}}$:

$$\mathbf{y}^{\{l\}} = c_m^{\{l\}} + \alpha V_m^{\{l\}} = \phi_{c-c}^{\{l\}}(c_m^{\{l-1\}}) + \alpha\phi_{c-V}^{\{l\}}(V_m^{\{l-1\}}) = \phi_c^{\{l\}}(\mathbf{y}^{\{l-1\}}) \implies \mathcal{Y}^{\{l\}} = \phi_c^{\{l\}}(\mathcal{Y}^{\{l-1\}}).$$

Therefore, the reachable set computation of a merged convolutional layer is defined in **Theorem 1**.

**Remark 2.** When $P(\alpha)$ is restricted to $C\alpha = d$, which means that the linear constraint yields only one feasible vector $\alpha$, the ImageStar set only has one vector $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$. In this special case, input the input reachable set into the merged convolutional layer,

$$\phi_c^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \phi_c^{\{l\}}(\mathbf{y}^{\{l-1\}}) = \phi_{c-c}^{\{l\}}(c_m^{\{l-1\}}) + \alpha\phi_{c-V}^{\{l\}}(V_m^{\{l-1\}}) = c_m^{\{l\}} + \alpha V_m^{\{l\}} = \mathbf{y}^{\{l\}} = \mathcal{Y}^{\{l\}}. \tag{11}$$

It shows that the merged convolutional layer yields the same corresponding output for the input reachable set no matter how many vectors are included in the set.

### 3.2.3. Merged fully connected layer

A conventional two-dimensional fully connected layer is following with parameters: the weights $\mathbf{W} \in \mathbb{R}^{n_{out} \times n_{in}}$, the biases $\mathbf{b} \in \mathbb{R}^{n_{out}}$, where $n_{out}$ is the number of output features and $n_{in}$ is the number of input features. Note that if the input is two or three-dimensional, it is required for dimension consistency between the weight matrix and the flattened input where $n_{in} = h \times w \times n_c$, $h, w, n_c$ are the height, width, and the number of channels of the input matrix respectively. $reshape()$ is the flattened operation for a higher dimension to one dimension. For two or three-dimensional input $\mathbf{x}$, $reshape()$ is applied first. $\overline{\mathbf{x}} = reshape(\mathbf{x}, n_{in})$, $\mathbf{x} \in \mathbb{R}^{h \times w \times nc}$, $\overline{\mathbf{x}} \in \mathbb{R}^{n_{in}}$. The fully connected layer operation $\phi_{fc}$ is the same as the linear operation represented below:

$$\mathbf{y} = \phi_{fc}(\overline{\mathbf{x}}) = \mathbf{W}\overline{\mathbf{x}} + \mathbf{b}. \tag{12}$$

Let $\mathcal{N}_1, \mathcal{N}_2$ be the two CNNs and consider the fully connected layer of the two CNNs is merged, the merged fully connected layer $\phi_{fc}(\cdot)$ and reachability computation are given in the following definition.

**Theorem 2.** *The merged fully connected layer $\phi_{fc}^{\{l\}}$ is defined as $\phi_{fc}^{\{l\}} = \langle \mathbf{W}_{1,fc}^{\{l\}}, \mathbf{b}_{1,fc}^{\{l\}}, \mathbf{W}_{2,fc}^{\{l\}}, \mathbf{b}_{2,fc}^{\{l\}} \rangle$, where $\mathbf{W}_{1,fc}^{\{l\}}, \mathbf{b}_{1,fc}^{\{l\}}$ are the weights and biases of network $\mathcal{N}_1$ at layer $l$, and $\mathbf{W}_{2,fc}^{\{l\}}, \mathbf{b}_{2,fc}^{\{l\}}$ are the weights and biases of network $\mathcal{N}_2$ at layer $l$. The merged weights and biases are reformatting as*

$$\mathbf{W}_m^{\{l\}} = \begin{bmatrix} \mathbf{W}_{1,fc}^{\{l\}} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{2,fc}^{\{l\}} \end{bmatrix}, \mathbf{b}_m^{\{l\}} = \begin{bmatrix} \mathbf{b}_{1,fc}^{\{l\}} \\ \mathbf{b}_{2,fc}^{\{l\}} \end{bmatrix}. \tag{13}$$

*The input set of layer $l$ is given as $\mathcal{Y}^{\{l-1\}} = \langle \overline{c}_m^{\{l-1\}}, \overline{V}_m^{\{l-1\}}, P_m^{\{l-1\}} \rangle$ and vector $\mathbf{y}^{\{l-1\}} \in \mathcal{Y}^{\{l-1\}}$ is a linear combination of the anchor image and generator images $\overline{\mathbf{y}}^{\{l-1\}} = \overline{c}_m^{\{l-1\}} + \alpha\overline{V}_m^{\{l-1\}}$.*

*The output reachable set $\mathcal{Y}^{\{l\}}$ is*

$$\mathcal{Y}^{\{l\}} = \phi_{fc}^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \langle \phi_{fc-c}^{\{l\}}(\overline{c}_m^{\{l-1\}}), \phi_{fc-V}^{\{l\}}(\overline{V}_m^{\{l-1\}}), \phi_{fc-P}^{\{l\}}(P_m^{\{l-1\}}) \rangle = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{fc}^{\{l\}}(\overline{\mathbf{y}}^{\{l-1\}})\}. \tag{14}$$

*$\phi_{fc-c}^{\{l\}}$ for anchor images $c_m$ is defined as:*

$$\phi_{fc-c}^{\{l\}}(\overline{c}_m^{\{l-1\}}) = \mathbf{W}_m^{\{l\}}\overline{c}_m^{\{l-1\}} + \mathbf{b}_m^{\{l\}} = \begin{bmatrix} \mathbf{W}_{1,fc}^{\{l\}} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{2,fc}^{\{l\}} \end{bmatrix} \begin{bmatrix} \overline{c}_1^{\{l-1\}} \\ \overline{c}_2^{\{l-1\}} \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{1,fc}^{\{l\}} \\ \mathbf{b}_{2,fc}^{\{l\}} \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{1,fc}^{\{l\}}\overline{c}_1^{\{l-1\}} + \mathbf{b}_{1,fc}^{\{l\}} \\ \mathbf{W}_{2,fc}^{\{l\}}\overline{c}_2^{\{l-1\}} + \mathbf{b}_{2,fc}^{\{l\}} \end{bmatrix}, \tag{15}$$

*where $\phi_{i,fc-c}^{\{l\}} = \langle \mathbf{W}_{i,fc}^{\{l\}}, \mathbf{b}_{i,fc}^{\{l\}} \rangle$ is the convolution operation with related parameters from network $\mathcal{N}_i$.*

*$\phi_{fc-V}^{\{l\}}$ for generator images $V_m$ is defined as:*

$$\phi_{fc-V}^{\{l\}}(\overline{V}_m^{\{l-1\}}) = \mathbf{W}_m^{\{l\}}\overline{V}_m^{\{l-1\}} = \begin{bmatrix} \mathbf{W}_{1,fc}^{\{l\}} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{2,fc}^{\{l\}} \end{bmatrix} \begin{bmatrix} \overline{V}_1^{\{l-1\}} \\ \overline{V}_2^{\{l-1\}} \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{1,fc}^{\{l\}}\overline{V}_1^{\{l-1\}} \\ \mathbf{W}_{2,fc}^{\{l\}}\overline{V}_2^{\{l-1\}} \end{bmatrix}, \tag{16}$$

*where $\phi_{i,fc-V}^{\{l\}} = \langle \mathbf{W}_{i,fc}^{\{l\}} \rangle$ is the convolution operation with related parameters from network $\mathcal{N}_i$.*

*$\phi_{fc-P}^{\{l\}}$ for linear constraint $P_m$ is defined as:*

$$\phi_{fc-P}^{\{l\}}(P_m^{\{l-1\}}) = P_m^{\{l-1\}}. \tag{17}$$

*Thus, the output reachable set $\mathcal{Y}^{\{l\}}$ can also be represented as:*

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{fc}^{\{l\}} \begin{bmatrix} \overline{\mathbf{y}}_1^{\{l-1\}} \\ \overline{\mathbf{y}}_2^{\{l-1\}} \end{bmatrix} \} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{1,fc}^{\{l\}}(\overline{\mathbf{y}}_1^{\{l-1\}}) \\ \phi_{2,fc}^{\{l\}}(\overline{\mathbf{y}}_2^{\{l-1\}}) \end{bmatrix} \}. \tag{18}$$

**Proof.** The anchor image $\overline{c}_m^{\{l\}} = Linear(\overline{c}_m^{\{l-1\}})$ where $Linear(\cdot)$ is the linear operation $\overline{c}_m^{\{l\}} = \mathbf{W}\overline{c}_m^{\{l-1\}} + \mathbf{b}$. If the anchor image is not a vector, it has to be flattened into a vector $\overline{c}_m^{\{l-1\}} = reshape(c_m^{\{l-1\}}, [n_{in}, 1])$. Thus, the anchor image of the merged fully connected layer is performed as below:

$$\begin{aligned} \phi_{fc-c}^{\{l\}}(\overline{c}_m^{\{l-1\}}) &= [\phi_{1,fc-c}^{\{l\}}(\overline{c}_1^{\{l-1\}})^T, \phi_{2,fc-c}^{\{l\}}(\overline{c}_2^{\{l-1\}})^T]^T = [(\mathbf{W}_{1,fc}^{\{l\}}\overline{c}_1^{\{l-1\}} + \mathbf{b}_{1,fc}^{\{l\}})^T, (\mathbf{W}_{2,fc}^{\{l\}}\overline{c}_2^{\{l-1\}} + \mathbf{b}_{2,fc}^{\{l\}})^T]^T = [(c_1^{\{l\}})^T, (c_2^{\{l\}})^T]^T \\ &= c_m^{\{l\}}. \end{aligned} \tag{19}$$

As for the generator images $\overline{V}_m^{\{l\}} = LinearZeroBias(\overline{V}_m^{\{l-1\}})$ where $LinearZeroBias(\cdot)$ is the linear operation with zero bias $\overline{V}_m^{\{l\}} = \mathbf{W}\overline{V}_m^{\{l-1\}}$. Generator images have to be flattened into a vector as the anchor image if they are not a vector $\overline{V}_m^{\{l-1\}} = reshape(V_m^{\{l-1\}}, [n_{in}, 1])$. Thus, the generator images of the merged fully connected layer are performed as below:

$$\phi_{fc-V}^{\{l\}}(\overline{V}_m^{\{l-1\}}) = [\phi_{1,fc-V}^{\{l\}}(\overline{V}_1^{\{l-1\}})^T, \phi_{2,fc-V}^{\{l\}}(\overline{V}_2^{\{l-1\}})^T]^T = [(\mathbf{W}_{1,fc}^{\{l\}}\overline{V}_1^{\{l-1\}})^T, (\mathbf{W}_{2,fc}^{\{l\}}\overline{V}_2^{\{l-1\}})^T]^T = [(V_1^{\{l\}})^T, (V_2^{\{l\}})^T]^T = V_m^{\{l\}}. \tag{20}$$

Additionally, the linear constraints are not changing $P_m^{\{l\}} = P_m^{\{l-1\}}$ because linear operation performs matrix multiplication and bias addition and has no effect on the constraint. Based on these, we can represent the vector $\mathbf{y}^{\{l\}}$ with the result $\mathbf{y}_1^{\{l\}}$, $\mathbf{y}_2^{\{l\}}$ from each network $\mathcal{N}_1$, $\mathcal{N}_2$:

$$\mathbf{y}^{\{l\}} = \phi_{fc-c}^{\{l\}}(\overline{c}_m^{\{l-1\}}) + \alpha\phi_{fc-V}^{\{l\}}(\overline{V}_m^{\{l-1\}}) = \begin{bmatrix} \phi_{1,fc-c}^{\{l\}}(\overline{c}_1^{\{l-1\}}) \\ \phi_{2,fc-c}^{\{l\}}(\overline{c}_2^{\{l-1\}}) \end{bmatrix} + \begin{bmatrix} \alpha\phi_{1,fc-V}^{\{l\}}(\overline{V}_1^{\{l-1\}}) \\ \alpha\phi_{2,fc-V}^{\{l\}}(\overline{V}_2^{\{l-1\}}) \end{bmatrix} = \begin{bmatrix} \phi_{1,fc}^{\{l\}}(\overline{\mathbf{y}}_1^{\{l-1\}}) \\ \phi_{2,fc}^{\{l\}}(\overline{\mathbf{y}}_2^{\{l-1\}}) \end{bmatrix}. \tag{21}$$

Thus, for each vector $\mathbf{y}^{\{l-1\}}$ in the input set $\mathcal{Y}^{\{l-1\}}$, it has a corresponding output vector $\mathbf{y}^{\{l\}}$:

$$\mathbf{y}^{\{l\}} = c_m^{\{l\}} + \alpha V_m^{\{l\}} = \phi_{fc-c}^{\{l\}}(\overline{c}_m^{\{l-1\}}) + \alpha\phi_{fc-V}^{\{l\}}(\overline{V}_m^{\{l-1\}}) = \phi_{fc}^{\{l\}}(\overline{\mathbf{y}}^{\{l-1\}}) \implies \mathcal{Y}^{\{l\}} = \phi_{fc}^{\{l\}}(\mathcal{Y}^{\{l-1\}}).$$

Therefore, the reachable set computation of a merged convolutional layer is defined in **Theorem 2**.

**Remark 3.** When $P(\alpha)$ is restricted to $C\alpha = d$, which means that the linear constraint yields only one feasible vector $\alpha$, the ImageStar set only has one vector $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$. In this special case, input the input reachable set into the merged fully connected layer,

$$\phi_{fc}^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \phi_{fc}^{\{l\}}(\overline{\mathbf{y}}^{\{l-1\}}) = \phi_{fc-c}^{\{l\}}(\overline{c}_m^{\{l-1\}}) + \alpha\phi_{fc-V}^{\{l\}}(\overline{V}_m^{\{l-1\}}) = c_m^{\{l\}} + \alpha V_m^{\{l\}} = \mathbf{y}^{\{l\}} = \mathcal{Y}^{\{l\}}. \tag{22}$$

It demonstrates that regardless of the number of vectors in the input reachable set, the merged fully connected layer yields the same output as the networks $\mathcal{N}_1$, $\mathcal{N}_2$ with the same input.

### 3.2.4. Merged max pooling layer

A conventional two-dimensional max pooling layer is followed with parameters: the pooling size $window(h, w) \in \mathbb{R}^{h\times w}$, the padding size $P$, and the stride $S$, where $h, w$ are the height and width of the pooling window. The max pooling operation is represented below:

$$\mathbf{y} = \phi_p(\mathbf{x}) = [\![\mathbf{x}]\!]_{window}, \tag{23}$$

where $[\![]\!]_{window}$ is the max pooling operation which finds and keeps the max value in the pooling range on the matrix, and the padding $P$ and stride $S$ affect the exact operation. Consider applying the max pooling operation to the two CNNs $\mathcal{N}_1, \mathcal{N}_2$ at the same layer, the merged max pooling layer $\phi_p(\cdot)$ and its reachability computation are given in the following definition.

**Theorem 3.** *The merged max pooling layer $\phi_p(\cdot)^{\{l\}}$ is defined as $\phi_p(\cdot)^{\{l\}} = \langle window_1^{\{l\}}, window_2^{\{l\}} \rangle$, where $window_1^{\{l\}}$ is the pooling size of network $\mathcal{N}_1$ at layer $l$, and $window_2^{\{l\}}$ is the pooling size of network $\mathcal{N}_2$ at layer $l$. The input set of layer $l$ is given as $\mathcal{Y}^{\{l-1\}} = \langle c_m^{\{l-1\}}, V_m^{\{l-1\}}, P_m^{\{l-1\}} \rangle$ and vector $\mathbf{y}^{\{l-1\}} \in \mathcal{Y}^{\{l-1\}}$ is a linear combination of the anchor image and generator images $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$.*

*The output reachable set $\mathcal{Y}^{\{l\}}$ is*

$$\mathcal{Y}^{\{l\}} = \phi_p^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = [\![c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}]\!]_{window}\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_p^{\{l\}}(\mathbf{y}^{\{l-1\}})\}, \tag{24}$$

*or in this representation*

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_p^{\{l\}} \begin{bmatrix} \mathbf{y}_1^{\{l-1\}} \\ \mathbf{y}_2^{\{l-1\}} \end{bmatrix} \} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{1,p}^{\{l\}}(\mathbf{y}_1^{\{l-1\}}) \\ \phi_{2,p}^{\{l\}}(\mathbf{y}_2^{\{l-1\}}) \end{bmatrix} \}. \tag{25}$$

*Depending on the linear constraint $P_m^{\{l-1\}}(\alpha)$, the reachable output set of the merger max pooling layer $\mathcal{Y}^{\{l\}}$ is the union of multiple outputs set $\mathcal{Y}^{\{l\}} = \{\mathcal{Y}_{m,1}^{\{l\}}, \mathcal{Y}_{m,2}^{\{l\}}, \ldots, \mathcal{Y}_{m,N_{set}}^{\{l\}}\}$, where each output set $\mathcal{Y}_{m,i}^{\{l\}}$ has distinct $\langle c_{m,i}^{\{l\}}, V_{m,i}^{\{l\}}, P_{m,i}^{\{l\}} \rangle$, and $N_{set}$ is the number of total output set after splitting.*

**Proof.** Reachability computation for the merged max pooling layer performs the same pooling operation as the normal max pooling layer except for the linear constraint part. Depending on the linear constraint $P_m$, if each region has and only has one max point position,

$$\begin{aligned} \phi_p^{\{l\}}(\mathcal{Y}^{\{l-1\}}) &= \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = [\![ c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}} ]\!]_{window} \} \\ &= \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = c_m^{\{l\}} + \alpha V_m^{\{l\}} \} \\ &= \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_p^{\{l\}}(\mathbf{y}^{\{l-1\}}) \} \\ &= \mathcal{Y}^{\{l\}}, \end{aligned}$$

where the value at the max point position in anchor image and generator images are retained in $c_m^{\{l\}}, V_m^{\{l\}}$ and other values are discarded, and linear constraints $P_m^{\{l-1\}} = P_m^{\{l\}}$ keep the same. Based on these, we can represent the vector $\mathbf{y}^{\{l\}}$ with the result $\mathbf{y}_1^{\{l\}}$, $\mathbf{y}_2^{\{l\}}$ from each network $\mathcal{N}_1$, $\mathcal{N}_2$:

$$\phi_p^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} [\![ c_1^{\{l-1\}} + \alpha V_1^{\{l-1\}} ]\!]_{window} \\ [\![ c_2^{\{l-1\}} + \alpha V_2^{\{l-1\}} ]\!]_{window} \end{bmatrix} \} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{1,p}^{\{l\}}(\mathbf{y}_1^{\{l-1\}}) \\ \phi_{2,p}^{\{l\}}(\mathbf{y}_2^{\{l-1\}}) \end{bmatrix} \}. \tag{26}$$

If one or more regions have multiple max point position candidates, each candidate will create a new reachable set by adding a new linear constraint, which means that the reachable set of the merged max pooling layer is the union of multiple distinct ImageStar reachable set $\mathcal{Y}_{m,1}^{\{l\}}, \mathcal{Y}_{m,2}^{\{l\}}, \ldots, \mathcal{Y}_{m,N_{set}}^{\{l\}}$. Though each $\mathcal{Y}_{m,i}^{\{l\}}$ have different $c_{m,i}^{\{l\}}, V_{m,i}^{\{l\}}, P_{m,i}^{\{l\}}$, each linear constraint is within the original linear constraint, which means that

$$P_m^{\{l-1\}} = P_{m,1}^{\{l-1\}} \cup P_{m,2}^{\{l-1\}} \cup \ldots \cup P_{m,N_{set}}^{\{l-1\}}.$$

In this case, $P_m^{\{l-1\}}$ can be split into many intervals $P_{m,i}^{\{l-1\}}$ to form input sets $\mathcal{Y}_{m,1}^{\{l-1\}}, \mathcal{Y}_{m,2}^{\{l-1\}}, \ldots, \mathcal{Y}_{m,N_{set}}^{\{l-1\}}$, and then we apply the max pooling operation for each input set. In each set, each region has only one max point position with its linear constraint $P_{m,i}^{\{l-1\}}$, which has been proved above. All split sets are included in the output set $\mathcal{Y}^{\{l-1\}} = \{\mathcal{Y}_{m,1}^{\{l-1\}}, \mathcal{Y}_{m,2}^{\{l-1\}}, \ldots, \mathcal{Y}_{m,N_{set}}^{\{l-1\}}\}$. Thus, the reachable set computation of a merged max pooling layer is defined in **Theorem 3**.

**Example 2.** To better demonstrate the operation of the merged max pooling layer, we give an example showing how to split the output reachable set depending on the linear constraints, as shown in Fig. 1.

The input is a $8 \times 4 \times 1$ matrix with a linear constraint $\alpha \in [-2, 2]$ applied on the position at $(1, 2)$. The merged max pooling layer uses a pool size of $2 \times 2$, a padding size of $P = [0, 0, 0, 0]$, and a stride $S = [2, 2]$. The pooling operation splits the matrix into eight regions $1, 2, 3, 4, 5, 6, 7, 8$, as shown in Fig. 1. Under the linear constraint $-2 \leq \alpha \leq 2$, regions $2, 3, 4, 5, 6, 7, 8$ have a distinct max point position which is the maximum value in that region. The merged max pooling layer performs the same operation in these regions as the normal max pooling layer. However, it is worth noticing that in the region 1, the max point position has two candidates, position $(1, 2)$ and position $(2, 2)$, which depends on the condition of $\alpha$. If $-1 \leq \alpha \leq 2$, the value at position $(1, 2)$ is $5 + \alpha \times 1 \implies 4 \leq 5 + \alpha \times 1 \leq 7$, which is always greater than the value $4 + \alpha \times 0$ at position $(2, 2)$. In this case, position $(1, 2)$ is the max point position. But if $-2 \leq \alpha \leq -1$, the value at position $(1, 2)$ is $3 \leq 5 + \alpha \times 1 \leq 4$, which is always smaller than the value $4 + \alpha \times 0$ at position $(2, 2)$. In this case, position $(2, 2)$ is the max point position.

Since the region 1 has two max point position candidates, the reachable set of the merged max pooling layer is the union of two new ImageStar, $\mathcal{Y}_{m,1}^{\{l\}}$ and $\mathcal{Y}_{m,2}^{\{l\}}$. In the first reachable set $\mathcal{Y}_{m,1}^{\{l\}}$, the max point position in region 1 is at $(1, 2)$ with a new linear constraint $-1 \leq \alpha \leq 2$. In the second reachable set $\mathcal{Y}_{m,2}^{\{l\}}$, the max point position in region 1 is at $(2, 2)$ with a new linear constraint $-2 \leq \alpha \leq -1$. Here, the reachable set of the merged max pooling layer has split the input reachable set into two new reachable sets.

Above is the illustration of the reachability computation of the merged max pooling layer. If more than one position has multiple maximum candidates, the split of the reachable sets will occur on each position, and It is worth noticing that, depending on the linear constraint and the matrix value, the number of reachable sets may grow exponentially.

$$Y_m^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}} = \begin{array}{|cc|cc|}\hline 0 & 5 & 0 & 5 \\ & 1 & & 2 \\ 1 & 4 & 1 & 3 \\ \hline 4 & 3 & 3 & 2 \\ & 3 & & 4 \\ 2 & 1 & 0 & 2 \\ \hline 1 & 1 & 1 & 0 \\ & 5 & & 6 \\ 2 & 4 & 3 & 0 \\ \hline 2 & 5 & 0 & 2 \\ & 7 & & 8 \\ 4 & 2 & 0 & 1 \\ \hline \end{array} + \alpha \begin{array}{|cc|cc|}\hline 0 & 1 & 0 & 0 \\ & 1 & & 2 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ & 3 & & 4 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ & 5 & & 6 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ & 7 & & 8 \\ 0 & 0 & 0 & 0 \\ \hline \end{array}, P_m^{\{l-1\}} \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

| Region | Max point position | Max point value | Condition |
|---|---|---|---|
| 2 | (1, 4) | $5 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 3 | (3, 1) | $4 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 4 | (3, 3) | $3 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 5 | (6, 2) | $4 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 6 | (6, 3) | $3 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 7 | (7, 2) | $5 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 8 | (7, 4) | $2 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| 1 | (1, 2) | $5 + \alpha * 1$ | $-1 \leq \alpha \leq 2$ |
| 1 | (2, 2) | $4 + \alpha * 0$ | $-2 \leq \alpha \leq -1$ |

$$Y_{m,1}^{\{l\}} = c_{m,1}^{\{l\}} + \alpha V_{m,1}^{\{l\}} = \begin{array}{|cc|}\hline 5 & 5 \\ 4 & 3 \\ 4 & 3 \\ 5 & 2 \\ \hline \end{array} + \alpha \begin{array}{|cc|}\hline 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \hline \end{array}, P_{m,1}^{\{l\}} \equiv \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

$$Y_{m,2}^{\{l\}} = c_{m,2}^{\{l\}} + \alpha V_{m,2}^{\{l\}} = \begin{array}{|cc|}\hline 4 & 5 \\ 4 & 3 \\ 4 & 3 \\ 5 & 2 \\ \hline \end{array} + \alpha \begin{array}{|cc|}\hline 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \hline \end{array}, P_{m,2}^{\{l\}} \equiv \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}$$

**Fig. 1.** Example of reachability computation of merged max pooling layer using ImageStars.

**Remark 4.** When $P(\alpha)$ is restricted to $C\alpha = d$, which means that the linear constraint yields only one feasible vector $\alpha$, the ImageStar set only has one vector $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$. In this special case, input the input reachable set into the merged max pooling layer,

$$\phi_p^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = [\![\mathbf{y}^{\{l-1\}}]\!]_{window} = [\![c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}]\!]_{window} = c_m^{\{l\}} + \alpha V_m^{\{l\}} = \mathbf{y}^{\{l\}} = \mathcal{Y}^{\{l\}}. \tag{27}$$

It shows that the merged max pooling layer yields the same corresponding output for the input reachable set no matter how many vectors are included in the set.

### 3.2.5. ReLU activation layer

The ReLU activation function is represented below:

$$ReLU(x) = max(0, x). \tag{28}$$

Consider applying the ReLU operation $\phi_{ReLU}(\cdot)$ to the two CNNs $\mathcal{N}_1, \mathcal{N}_2$ at the same layer level, the ReLU activation layer and reachability computation of the ReLU layer is given in the following definition.

**Theorem 4.** *The ReLU layer $\phi_{ReLU}^{\{l\}}(\cdot)$ with a reachable input set can be performed as a sequence of ReLU operations over all elements of the input set. The input set of layer $l$ is given as $\mathcal{Y}^{\{l-1\}} = \langle c_m^{\{l-1\}}, V_m^{\{l-1\}}, P_m^{\{l-1\}} \rangle$ and the reachability computation for the input is performed on each element in the input vector $\mathbf{y}^{\{l-1\}} \in \mathcal{Y}^{\{l-1\}}$ where $\mathbf{y}^{\{l-1\}}$ is a linear combination of the anchor image and generator images $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$.*

*For output vector* $\mathbf{y}^{\{l\}}$*:*

$$\mathbf{y}^{\{l\}} = ReLU_N(ReLU_{N-1}(...(ReLU_1(y_i^{\{l-1\}})))) \textbf{ where } ReLU_i(y_i^{\{l-1\}}) = \begin{cases} y_i^{\{l-1\}} & , y_i^{\{l-1\}} \geq 0 \\ 0 & , y_i^{\{l-1\}} < 0 \end{cases}, \tag{29}$$

*where* $N$ *is the total number of elements in the input vector* $\mathbf{y}^{\{l-1\}}$ *and* $y_i^{\{l-1\}}$ *is the* $i^{th}$ *element of the input vector. The* $ReLU_i$ *operation performs* $max(0, y_i^{\{l-1\}})$ *on the* $i^{th}$ *element and determines whether split a new output set at the* $i^{th}$ *element depending on the linear constraint. It remains the value if the* $i^{th}$ *element is greater than zero or we reset the* $i^{th}$ *element to zero if it is smaller than zero. The reachable output set of the ReLU activation layer* $\mathcal{Y}^{\{l\}}$ *is the union of multiple output sets* $\mathcal{Y}^{\{l\}} = \{\mathcal{Y}_{m,1}^{\{l\}}, \mathcal{Y}_{m,2}^{\{l\}}, \dots, \mathcal{Y}_{m,N_{set}}^{\{l\}}\}$*, where each output set* $\mathcal{Y}_{m,i}^{\{l\}}$ *has distinct* $\langle c_{m,i}^{\{l\}}, V_{m,i}^{\{l\}}, P_{m,i}^{\{l\}} \rangle$*, and* $N_{set}$ *is the number of total outputs set after splitting. The output* $\mathbf{y}^{\{l\}}$ *is a linear combination of the anchor image and generator images* $\mathbf{y}^{\{l\}} = c_m^{\{l\}} + \alpha V_m^{\{l\}}$*. We define* $\phi_{ReLU}^{\{l\}}(\cdot) = ReLU_N(ReLU_{N-1}(...(ReLU_1(\cdot))))$*. Thus, we have*

$$\mathcal{Y}^{\{l\}} = \phi_{ReLU}^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{ReLU}^{\{l\}}(\mathbf{y}^{\{l-1\}})\}, \tag{30}$$

*or in this representation*

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{ReLU}^{\{l\}} \begin{bmatrix} \mathbf{y}_1^{\{l-1\}} \\ \mathbf{y}_2^{\{l-1\}} \end{bmatrix}\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{ReLU}^{\{l\}}(\mathbf{y}_1^{\{l-1\}}) \\ \phi_{ReLU}^{\{l\}}(\mathbf{y}_2^{\{l-1\}}) \end{bmatrix}\}. \tag{31}$$

**Proof.** Reachability computation for the ReLU activation layer performs a sequence of $max(0, y_i)$ function on each element in the reachable input set. Depending on the linear constraint, if an element is always negative, it is reset to zero.

$$y_i^{\{l\}} = c_{m_i}^{\{l\}} + \alpha V_{m_i}^{\{l\}} = 0 \implies c_{m_i}^{\{l\}}, V_{m_i}^{\{l\}} = 0, P_m^{\{l-1\}} = P_m^{\{l\}}, \text{ if } y_i^{\{l-1\}} < 0.$$

If an element is always positive, it remains the same.

$$y_i^{\{l\}} = c_{m_i}^{\{l\}} + \alpha V_{m_i}^{\{l\}} = y_i^{\{l-1\}} \implies c_{m_i}^{\{l\}} = c_{m_i}^{\{l-1\}}, V_{m_i}^{\{l\}} = V_{m_i}^{\{l-1\}}, P_m^{\{l-1\}} = P_m^{\{l\}}, \text{ if } y_i^{\{l-1\}} \geq 0.$$

If an element may be negative or positive with an extra constraint, reachability computation for ReLU activation splits the input set into two new reachable sets with new linear constraints, which means that the reachable set of the ReLU activation layer is the union of multiple distinct ImageStar reachable set $\mathcal{Y}_{m,1}^{\{l\}}, \mathcal{Y}_{m,2}^{\{l\}}, \dots, \mathcal{Y}_{m,N_{set}}^{\{l\}}$. For each reachable set, the linear constraint is an interval within the original linear constraint, which means that

$$P_m^{\{l-1\}} = P_{m,1}^{\{l-1\}} \cup P_{m,2}^{\{l-1\}} \cup \dots \cup P_{m,N_{set}}^{\{l-1\}}.$$

Because ReLU activation is performed on each element, we can represent the vector $\mathbf{y}^{\{l\}}$ with the result $\mathbf{y}_1^{\{l\}}$, $\mathbf{y}_2^{\{l\}}$ from each network $\mathcal{N}_1$, $\mathcal{N}_2$ and perform ReLU activation on each vector $\mathbf{y}_1^{\{l\}}$, $\mathbf{y}_2^{\{l\}}$:

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{ReLU}^{\{l\}}(\mathbf{y}^{\{l-1\}})\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{ReLU}^{\{l\}} \begin{bmatrix} \mathbf{y}_1^{\{l-1\}} \\ \mathbf{y}_2^{\{l-1\}} \end{bmatrix}\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{ReLU}^{\{l\}}(\mathbf{y}_1^{\{l-1\}}) \\ \phi_{ReLU}^{\{l\}}(\mathbf{y}_2^{\{l-1\}}) \end{bmatrix}\}. \tag{32}$$

Same as the merged max pooling layer, the original linear constraint $P_m^{\{l-1\}}$ can be split into many intervals $P_{m,i}^{\{l-1\}}$ to form input sets $\mathcal{Y}_{m,1}^{\{l-1\}}, \mathcal{Y}_{m,2}^{\{l-1\}}, \dots, \mathcal{Y}_{m,N_{set}}^{\{l-1\}}$, and then we apply the ReLU function for each input set. In each set, each element is always positive or negative with its linear constraint $P_{m,i}^{\{l-1\}}$, and their reachability computation has been proved above. Therefore, the reachable set computation of a ReLU activation layer is defined in **Theorem 4**.

**Example 3.** To better demonstrate the operation of the ReLU activation layer, we give an example showing how to split the output reachable set depending on the linear constraint, as shown in Fig. 2.

The input is $4 \times 2 \times 1$ matrix with a linear constraint $\alpha \in [-2, 2]$ applied on the position at $(1, 1)$. A sequence of ReLU functions is applied to each element in the input matrix, as shown in Fig. 2. Under the linear constraint $-2 \leq \alpha \leq 2$, elements at positions $(1,2), (2,1), (2,2), (3,1), (3,2), (4,1), (4,2)$ are discriminated. The ReLU function doesn't split the reachable set for these elements. However, it is interesting to observe that the element at position $(1, 1)$ has a range across the zero point. If $-2 \leq \alpha \leq 1$, the element value is $-1 + \alpha * 1 \leq 0$, meaning the element should be reset to zero. If $1 \leq \alpha \leq 2$, the element value is $-1 + \alpha * 1 \geq 0$, meaning the element should be retained.

Since the ReLU function outputs yield two different results, the ReLU activation produces two new ImageStar, $\mathcal{Y}_{m,1}^{\{l\}}$ and $\mathcal{Y}_{m,2}^{\{l\}}$. In the first reachable set $\mathcal{Y}_{m,1}^{\{l\}}$, the linear constraint is $-2 \leq \alpha \leq 1$. In the second reachable set $\mathcal{Y}_{m,2}^{\{l\}}$, the linear constraint is $1 \leq \alpha \leq 2$. The union of $\mathcal{Y}_{m,1}^{\{l\}}$ and $\mathcal{Y}_{m,2}^{\{l\}}$ is the output reachable sets of the ReLU activation layer.

Above is the illustration of the reachability computation of the ReLU activation layer. Same as in Example 2, the split will occur on each element if it is not determined, and the number of reachable sets may grow exponentially.

$$Y_m^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}} = \begin{bmatrix} -1 & 1 \\ 2 & 0 \\ -1 & -2 \\ 0 & 3 \end{bmatrix} + \alpha \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, P_m^{\{l-1\}} \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

| Position | Current value | ReLU value | Condition |
|---|---|---|---|
| (1,2) | $1 + \alpha * 0$ | $1 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (2,1) | $2 + \alpha * 0$ | $2 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (2,2) | $0 + \alpha * 0$ | $0 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (3,1) | $-1 + \alpha * 0$ | $0 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (3,2) | $-2 + \alpha * 0$ | $0 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (4,1) | $0 + \alpha * 0$ | $0 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (4,2) | $3 + \alpha * 0$ | $3 + \alpha * 0$ | $-2 \leq \alpha \leq 2$ |
| (1,1) | $-1 + \alpha * 1$ | $0 + \alpha * 0$ | $-2 \leq \alpha \leq 1$ |
| (1,1) | $-1 + \alpha * 1$ | $-1 + \alpha * 1$ | $1 \leq \alpha \leq 2$ |

$$Y_{m,1}^{\{l\}} = c_{m,1}^{\{l\}} + \alpha V_{m,1}^{\{l\}} = \begin{bmatrix} 0 & 1 \\ 2 & 0 \\ 0 & 0 \\ 0 & 3 \end{bmatrix} + \alpha \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, P_{m,1}^{\{l\}} \equiv \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$$

$$Y_{m,2}^{\{l\}} = c_{m,2}^{\{l\}} + \alpha V_{m,2}^{\{l\}} = \begin{bmatrix} -1 & 1 \\ 2 & 0 \\ 0 & 0 \\ 0 & 3 \end{bmatrix} + \alpha \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, P_{m,2}^{\{l\}} \equiv \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}$$

**Fig. 2.** Reachability computation of ReLU layer using ImageStars.

**Remark 5.** When $P(\alpha)$ is restricted to $C\alpha = d$, which means that the linear constraint yields only one feasible vector $\alpha$, the ImageStar set only has one vector $\mathbf{y}^{\{l-1\}} = c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}$. In this special case, input the input reachable set into the merged ReLU layer,

$$\phi_p^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \phi_{ReLU}^{\{l\}}(\mathbf{y}^{\{l-1\}}) = \phi_{ReLU}^{\{l\}}(c_m^{\{l-1\}} + \alpha V_m^{\{l-1\}}) = c_m^{\{l\}} + \alpha V_m^{\{l\}} = \mathbf{y}^{\{l\}} = \mathcal{Y}^{\{l\}}. \tag{33}$$

It shows that the merged ReLU layer yields the same corresponding output for the input reachable set no matter how many vectors are included in the set.

### 3.2.6. Comparison layer

After computing the reachable set through all layers in $\mathcal{N}_1, \mathcal{N}_2$, an extra linear layer is added for computing the reachable set of the difference between two outputs. The comparison layer is defined as below:

$$\mathbf{y} = \phi_{cmp}(\mathbf{x}) = [\mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}}]\mathbf{x}, \tag{34}$$

where $\mathbf{I}_{n_{in}} \in \mathbb{R}^{n_{in} \times n_{in}}$ represents the identity matrix and $n_{in}$ is the number of input features. Let $\mathcal{N}_1, \mathcal{N}_2$ be the two CNNs and consider adding an extra linear layer after the computation of the output reachable set from the merged network, the comparison layer $\phi_{cmp}(\cdot)$ and reachability computation of the difference between the two CNNs $\mathcal{N}_1, \mathcal{N}_2$ is given in the following definition.

**Theorem 5.** *The comparison layer $\phi_{cmp}^{\{L+1\}}$ is defined as $\phi_{cmp}^{\{L+1\}} = \langle \mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}} \rangle$, where $n_{in}$ is the dimension of the input and $\mathbf{I}_{n_{in}}$ is a $n_{in}$ dimensional identity matrix. The merged weights are reformatting as $\mathbf{W}_m^{\{L+1\}} = [\mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}}]$. The input set of layer $L+1$ is given as $\mathcal{Y}^{\{L\}} = \langle c_m^{\{L\}}, V_m^{\{L\}}, P_m^{\{L\}} \rangle$ and vector $\mathbf{y}^{\{L\}} \in \mathcal{Y}^{\{L\}}$ is a linear combination of the anchor image and generator images $\mathbf{y}^{\{L\}} = c_m^{\{L\}} + \alpha V_m^{\{L\}}$. The output reachable set $\mathcal{Y}^{\{L+1\}}$ is*

$$\mathcal{Y}^{\{L+1\}} = \phi_{cmp}^{\{L+1\}}(\mathcal{Y}^{\{L\}}) = \langle \phi_{cmp-c}^{\{L+1\}}(c_m^{\{L\}}), \phi_{cmp-V}^{\{L+1\}}(V_m^{\{L\}}), \phi_{cmp-P}^{\{L+1\}}(P_m^{\{L\}}) \rangle. \tag{35}$$

$\phi_{cmp-c}^{\{L+1\}}$ *for anchor image* $c_m$ *is defined as:*

$$\phi_{cmp}^{\{L+1\}}(c_m^{\{L\}}) = \mathbf{W}_m^{\{L+1\}} c_m^{\{L\}} = \left[\mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}}\right] \begin{bmatrix} c_{m,1}^{\{L\}} \\ c_{m,2}^{\{L\}} \end{bmatrix} = \mathbf{I}_{n_{in}} c_{m,1}^{\{L\}} - \mathbf{I}_{n_{in}} c_{m,2}^{\{L\}} = c_{m,1}^{\{L\}} - c_{m,1}^{\{L\}}. \tag{36}$$

$\phi_{cmp-V}^{\{L+1\}}$ *for generator images* $V_m$ *is defined as:*

$$\phi_{cmp}^{\{L+1\}}(V_m^{\{L\}}) = \mathbf{W}_m^{\{L+1\}} V_m^{\{L\}} = \left[\mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}}\right] \begin{bmatrix} V_{m,1}^{\{L\}} \\ V_{m,2}^{\{L\}} \end{bmatrix} = \mathbf{I}_{n_{in}} V_{m,1}^{\{L\}} - \mathbf{I}_{n_{in}} V_{m,2}^{\{L\}} = V_{m,1}^{\{L\}} - V_{m,1}^{\{L\}}. \tag{37}$$

*Thus, the output reachable set* $\mathcal{Y}^{\{L+1\}}$ *is the set of differences between the output from two networks* $\mathcal{N}_1$, $\mathcal{N}_2$,

$$\mathcal{Y}^{\{L+1\}} = \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \phi_{cmp-c}^{\{L+1\}}(c_m^{\{L\}}) + \alpha\phi_{cmp-V}^{\{L+1\}}(V_m^{\{L\}})\} = \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \mathbf{y}_1^{\{L\}} - \mathbf{y}_2^{\{L\}}\}. \tag{38}$$

**Proof.** Considering an input reachable set $\mathcal{X} = \langle c_m^{\{0\}}, V_m^{\{0\}}, P_m^{\{0\}} \rangle$ and input vector $\mathbf{x}^{\{0\}} \in \mathcal{X}$. Normally, the first layer of the convolutional neural network is a convolutional layer. The results for the output of the convolutional layer of merged neural network $\mathcal{N}_M$ can be obtained by following **Theorem 1**

$$\mathcal{Y}^{\{1\}} = \phi_c^{\{1\}}(\mathcal{X}^{\{0\}}) = \langle \phi_{c-c}^{\{1\}}(c_m^{\{0\}}), \phi_{c-V}^{\{1\}}(V_m^{\{0\}}), \phi_{c-P}^{\{1\}}(P_m^{\{0\}}) \rangle = \{\mathbf{y}^{\{1\}} | \mathbf{y}^{\{1\}} = \phi_c^{\{1\}}(\mathbf{x}^{\{0\}})\} = \{\mathbf{y}^{\{1\}} | \mathbf{y}^{\{1\}} = \begin{bmatrix} \phi_{1,c}^{\{1\}}(\mathbf{x}_1^{\{0\}}) \\ \phi_{2,c}^{\{1\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix}\}. \tag{39}$$

Then, considering the convolutional layer is followed by a ReLU activation layer, based on **Theorem 4**, it leads to

$$\mathcal{Y}^{\{2\}} = \phi_{ReLU}^{\{2\}}(\mathcal{Y}^{\{1\}}) = \{\mathbf{y}^{\{2\}} | \mathbf{y}^{\{2\}} = \phi_{ReLU}^{\{2\}}(\mathbf{y}^{\{1\}})\}. \tag{40}$$

Recursively, we can obtain

$$\mathcal{Y}^{\{2\}} = \{\mathbf{y}^{\{2\}} | \mathbf{y}^{\{2\}} = \phi_{ReLU}^{\{2\}} \circ \phi_c^{\{1\}}(\mathbf{x}^{\{0\}})\} = \{\mathbf{y}^{\{2\}} | \mathbf{y}^{\{2\}} = \begin{bmatrix} \phi_{ReLU}^{\{2\}} \circ \phi_{1,c}^{\{1\}}(\mathbf{x}_1^{\{0\}}) \\ \phi_{ReLU}^{\{2\}} \circ \phi_{2,c}^{\{1\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix}\}. \tag{41}$$

After a few combinations of the convolutional layer and ReLU layer, a max pooling layer is performed, and we can obtain the results for the output of the max pooling layer following **Theorem 3**

$$\mathcal{Y}^{\{l\}} = \phi_p^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_p^{\{l\}}(\mathbf{y}^{\{l-1\}})\}, \tag{42}$$

which yields

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_p^{\{l\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_c^{\{1\}}(\mathbf{x}^{\{0\}})\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{1,p}^{\{l\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_{1,c}^{\{1\}}(\mathbf{x}_1^{\{0\}}) \\ \phi_{2,p}^{\{l\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_{2,c}^{\{1\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix}\}. \tag{43}$$

Further, considering all convolution and pooling operations have finished, the merged neural network is on a fully connected layer. According to **Theorem 2**, it leads to

$$\mathcal{Y}^{\{l\}} = \phi_{fc}^{\{l\}}(\mathcal{Y}^{\{l-1\}}) = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{fc}^{\{l\}}(\mathbf{y}^{\{l-1\}})\}, \tag{44}$$

and we can obtain that

$$\mathcal{Y}^{\{l\}} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \phi_{fc}^{\{l\}} \circ \phi_p^{\{l-1\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_c^{\{1\}}(\mathbf{x}^{\{0\}})\} = \{\mathbf{y}^{\{l\}} | \mathbf{y}^{\{l\}} = \begin{bmatrix} \phi_{1,fc}^{\{l\}} \circ \phi_{1,p}^{\{l-1\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_{1,c}^{\{1\}}(\mathbf{x}_1^{\{0\}}) \\ \phi_{2,fc}^{\{l\}} \circ \phi_{2,p}^{\{l-1\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_{2,c}^{\{1\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix}\}. \tag{45}$$

Moreover, after a few groups of fully connected layer and ReLU layer, all layers in the original CNNs $\mathcal{N}_1, \mathcal{N}_2$ have finished, and the results for output of the two CNNs can be obtained

$$\begin{aligned} \mathcal{Y}^{\{L\}} &= \{\mathbf{y}^{\{L\}} | \mathbf{y}^{\{L\}} = \phi_{ReLU}^{\{L\}} \circ \phi_{fc}^{\{L-1\}} \circ \phi_{ReLU}^{\{L-2\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_c^{\{1\}}(\mathbf{x}^{\{0\}})\} \\ &= \{\mathbf{y}^{\{L\}} | \mathbf{y}^{\{L\}} = \begin{bmatrix} \phi_{ReLU}^{\{L\}} \circ \phi_{1,fc}^{\{L-1\}} \circ \phi_{ReLU}^{\{L-2\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_{1,c}^{\{1\}}(\mathbf{x}_1^{\{0\}}) \\ \phi_{ReLU}^{\{L\}} \circ \phi_{2,fc}^{\{L-1\}} \circ \phi_{ReLU}^{\{L-2\}} \circ \dots \phi_{ReLU}^{\{2\}} \circ \phi_{2,c}^{\{1\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix}\} \\ &= \{\mathbf{y}^{\{L\}} | \mathbf{y}^{\{L\}} = \begin{bmatrix} \Phi_1^{\{L\}}(\mathbf{x}_1^{\{0\}}) \\ \Phi_2^{\{L\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix}\}, \end{aligned}$$

where $\Phi_i^{\{L\}}$ is the mapping relation of the CNN $\mathcal{N}_i$ which includes all layer operation in $\mathcal{N}_i$ and follows the sequence of layer operation in $\mathcal{N}_i$ from top to bottom with $\phi_i^{\{1\}}, \phi_i^{\{2\}}, \dots, \phi_i^{\{L\}}$. Thus, the output $\mathbf{y}_i^{\{L\}}$ of each network $\mathcal{N}_i$ can be represented by the input $\mathbf{x}_i^{\{0\}}$ and its related mapping relation $\Phi_i^{\{L\}}$

$$\mathbf{y}_1^{\{L\}} = \Phi_1^{\{L\}}(\mathbf{x}_1^{\{0\}}) \, , \, \mathbf{y}_2^{\{L\}} = \Phi_2^{\{L\}}(\mathbf{x}_2^{\{0\}}). \tag{46}$$

At last, the difference of the output $\mathbf{y}_1^{\{L\}}$ of $\mathcal{N}_1$ and the output $\mathbf{y}_2^{\{L\}}$ of $\mathcal{N}_2$ can be obtained

$$\begin{aligned}
\mathcal{Y}^{\{L+1\}} &= \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \phi_{cmp}^{\{L+1\}}(\mathbf{y}^{\{L\}})\} \\
&= \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \begin{bmatrix} \mathbf{I}_{n_{in}}, -\mathbf{I}_{n_{in}} \end{bmatrix} \begin{bmatrix} \Phi_1^{\{L\}}(\mathbf{x}_1^{\{0\}}) \\ \Phi_2^{\{L\}}(\mathbf{x}_2^{\{0\}}) \end{bmatrix} \} \\
&= \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \mathbf{I}_{n_{in}} \Phi_1^{\{L\}}(\mathbf{x}_1^{\{0\}}) - \mathbf{I}_{n_{in}} \Phi_2^{\{L\}}(\mathbf{x}_2^{\{0\}})\} \\
&= \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \mathbf{y}_1^{\{L\}} - \mathbf{y}_2^{\{L\}}\}.
\end{aligned}$$

Therefore, $\mathcal{Y}^{\{L+1\}}$ is the reachable set of differences between the output from two networks $\mathcal{N}_1$, $\mathcal{N}_2$. The proof is complete.

### 3.3. Maximum discrepancy computation algorithm

The detailed procedure of our algorithm is summarized in Algorithm 1.

**Algorithm 1** Maximum discrepancy computation.

**Require:**
Original Convolutional Neural Network $\mathcal{N}_1$
Compressed Convolutional Neural Network $\mathcal{N}_2$
Target Input $\mathbf{X}$ and its lower and upper bound $\mathbf{L}, \mathbf{U}$

Compute reachable set of input $\mathcal{X}$ from $\mathbf{X}, \mathbf{L}, \mathbf{U}$ using *ImageStar*
Merge two networks into one $\mathcal{N}_m$
**for** layer = 1 to $L$ **do**
  **for** each reachable set **do**
    Compute reachable set output of the layer $\phi_m^{\{layer\}}(\cdot)$
  **end for**
**end for**
**for** each reachable set **do**
  Compute output difference $\mathcal{Y}^{\{L+1\}} = \{\mathbf{y}^{\{L+1\}} | \mathbf{y}^{\{L+1\}} = \mathbf{y}_1^{\{L\}} - \mathbf{y}_2^{\{L\}}\}$
**end for**
**return** $\mathcal{Y}^{\{L+1\}}$

## 4. Experiments

The evaluation of our approach consists of two parts. First, we verify the effectiveness of the compression error computation by performing a numerical example. Second, we give a real-world classification problem using the Cifar10 dataset and state-of-the-art compression method to examine the compression performance and characterize the discrepancy between the original and the compressed one. The experiments are done on a computer with the following configurations: AMD Ryzen 7 5800X 8-Core Processor @ 3.8 GHz, 32 GB Memory, NVIDIA GeForce RTX 3800 Ti, Windows 11 Pro OS. The source code is fully available online.[2]

### 4.1. Effectiveness verification of compression error computation

We verify the effectiveness of our compression error computation method by using a numerical experiment. In this numerical experiment, we aim to soundly simulate two neural networks containing two convolution layers, one max pooling layer, and one fully connected layer. Each convolution layer has an activation function of ReLU. The input of the networks is a $6 \times 6 \times 1$ matrix. To facilitate the visualization of the simulation results, the networks' outputs are selected as one-dimensional.

First, we randomly generate two convolutional neural networks $\mathcal{N}_1, \mathcal{N}_2$ where the weights and biases are randomly generated following the Gaussian distribution with a mean of zero and standard deviation of one. Second, a merged network $\mathcal{N}_m$ is constructed from $\mathcal{N}_1$ and $\mathcal{N}_2$ following the definition in Section 3. Third, the input matrix is randomly generated with a lower and upper bound where the bounds are all zero except the top left $2 \times 2$ corner with a 0.1 bound range (lower bound is -0.05 and upper bound is 0.05 at the corner). After the reachability computation, the error range with a lower and upper bound is obtained. With the help of the error range, the upper bound and lower bound of output $\mathbf{y}_2^{\{L\}}$ of $\mathcal{N}_2$ can be obtained via the output $\mathbf{y}_1^{\{L\}}$ of $\mathcal{N}_1$. To verify the bound is the assured output range for $\mathcal{N}_2$, random noise within the bound is generated and applied on the input multiple times and records their corresponding output.

Output data of network $\mathcal{N}_1$ and the output range for network $\mathcal{N}_2$, as well as the output of $\mathcal{N}_2$ with random noise, are presented in Fig. 3. The error range can be obtained from Eq. (38). In this case, ten random inputs have been generated, and each index represents a random input. The blue line represents the upper output range of the network $\mathcal{N}_2$, and the green line represents the lower output

[2] The source code for experiments is publicly available: https://github.com/aicpslab/merged-cnn.

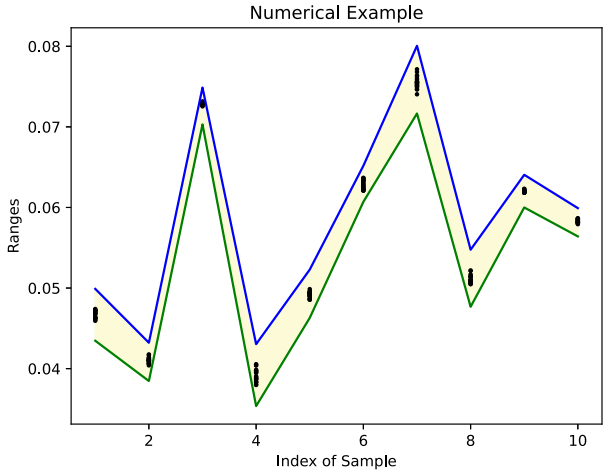


**Fig. 3.** Assured output range for the second random convolutional neural network ($\mathcal{N}_2$) by error computation method. The blue line is the upper bound, and the green line is the lower bound. The yellow area is the guaranteed output range. Random black dot outputs all within the area.

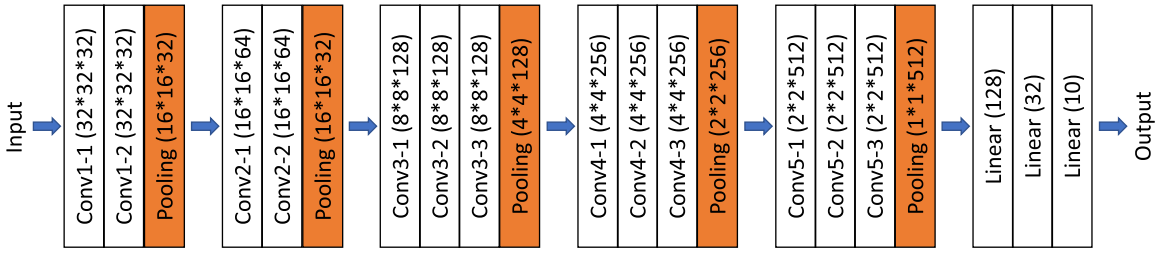


**Fig. 4.** VGG model configuration. It has 16 layers with trainable parameters. The number in parentheses is the dimension of the matrix after layer computation with the order height∗width∗channel.

range of $\mathcal{N}_2$. The yellow area, bounded by the blue line and the green line, is the guaranteed output area of the network $\mathcal{N}_2$. Black dots represent the output of $\mathcal{N}_2$ with bounded random noise on the input. We randomly generate 20 noised inputs for each sample and calculate the output of $\mathcal{N}_2$. It can be observed that all noised inputs are within the output range. Besides, the output range varies with different random ranges. Index 4 input has an output range of around 0.01. However, the index six input only has an output range smaller than 0.005. It explains that, for different inputs, the output discrepancy between the two networks is different.

### *4.2. Discrepancy evaluation of quantization method on Cifar10 dataset*

To evaluate the performance of a state-of-the-art compression method on a real-world classification network, we train a VGG16 model and its quantization version with 80% and 81% accuracy, respectively. The Cifar10 data set consists of 60,000 images with a resolution of $32 \times 32$ pixels color images in 10 different classes [15]. The network architectures are shown in Fig. 4. For the quantization network, we apply the quantization aware training (QAT) method embedded in Pytorch [14] on the original VGG16 network. The QAT method retrains the model's parameters, simulates the effect of quantization during training, and quantizes the weights and activation after conversion, which explains why the quantization network has higher accuracy than the original one. The number of parameters and size of the two networks are shown in Table 1. It is interesting to note that the two networks have the same number of parameters. Still, the quantization network is four times smaller than the original network, proving that the quantization method helps narrow the size of any large network.

In this case experiment, first, we load the two networks and construct the merged network. Second, we give a test image and define the small perturbation on the top left corner with a 0.1 bound range (lower bound is −0.05 and upper bound is 0.05). Third, we build the input ImageStar set with the input image, lower and upper bound, and input it to the merged network with our computation method. The network classifies the image into ten classes: airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck. Each class has an output to indicate the score of each label, and the label with the highest score gives the prediction result of the image. Thus, our method will compute the difference between the two networks and give each class an error range. With the help of the error range, the predicted output range of the quantization network can be computed by the original network and error range of each class.

The input image of the merged network is an airplane, and the noise perturbation is in the top left corner. The original network's output data and the quantization network's output range are presented in Fig. 5. The row index represents each label, and the value indicates the prediction score for each label. The blue dots are the scoring output of the original network, and it shows that the

**Table 1**
Parameters and size of the two networks.

| Network | Number of Parameters (in millions) | Size (MB) | Accuracy (%) |
|---|---|---|---|
| Original Network | 7.9 | 30.0 | 80% |
| Quantization Network | 7.9 | 7.6 | 81% |

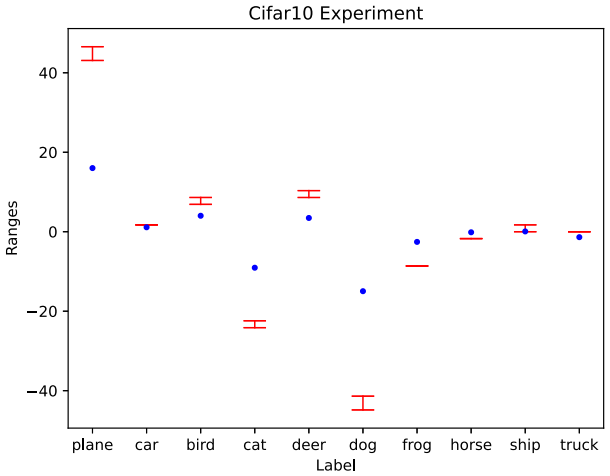


**Fig. 5.** Original network output and guarantee output range of quantization network. Blue dots are the output for the original network. The red whisker represents the output range of the quantization network computed by the error computation method.

**Table 2**
Output detail for original network and quantization network. Ori. is the output of the original network. The lower bound (LB) and upper bound (UB) are the output range for the quantization network.

| Class | Airplane | Car | Bird | Cat | Deer | Dog | Frog | Horse | Ship | Truck |
|---|---|---|---|---|---|---|---|---|---|---|
| Ori. | 16.0246 | 1.1289 | 4.0271 | −9.0505 | 3.4717 | −14.9569 | −2.548 | −0.1301 | 0.0852 | −1.3598 |
| LB | 43.1027 | 1.7212 | 6.8931 | −24.1441 | 8.6207 | −44.8416 | −8.6233 | −1.7290 | −0.0049 | −0.0063 |
| UB | 46.5671 | 1.7255 | 8.6246 | −22.4113 | 10.3497 | −41.3786 | −8.6177 | −1.7187 | 1.7281 | 0.0063 |

airplane class has the highest score, which means that the original network will give airplane prediction output for this image. The red whisker on each label is the output range for the quantization network, and the detail is shown in Table 2. The output range of the quantization network could be higher or lower than the output of the original network, which is hard to track the error direction. Considering the QAT method is a retrained-like quantization method, it may change the distribution of parameters in each layer instead of directly quantizing the parameters. Interestingly, the airplane class still has the highest output range among the ten classes, which means that the quantization network has robustness classification capability on the input image even with the perturbation in the bound. But it is also possible that the class with the highest output range is different from the ground truth class, or there is an overlapping range for multiple classes with high scores. In these cases, the quantization network may give wrong predictions with the perturbation effect on the input image, which indicates that the quantization network needs to be more robust compared to the original network.

However, from Table 2, we notice that the discrepancy between the two networks is significant, especially at the airplane, cat, and dog labels. The discrepancies are all over 20, which means that their outputs have significant differences at these labels. This result indicates that the QAT quantization method leads to huge differences between the compressed network and the original one, where the compressed network can't retain the output characteristics of the original network. The QAT quantization method is not suitable for use in safety-critical applications.

For this experiment case, the computation time for the compression network guaranteed output range is about 20 minutes. However, according to Example 2 and 3, the split operation leads to an exponential growth of reachable sets, significantly affecting the computational time of the whole reachability computation. If the two networks are not close, the discrepancy would be huge, and the total computation time could be over an hour, where we have met the situation during our experiment test. Besides, the computational time can be varied based on the input. If the two networks give different predictions of an input image, the guaranteed output error computation will take a long time to finish, from an hour to ten hours, or even stop because of running out of memory during computation.

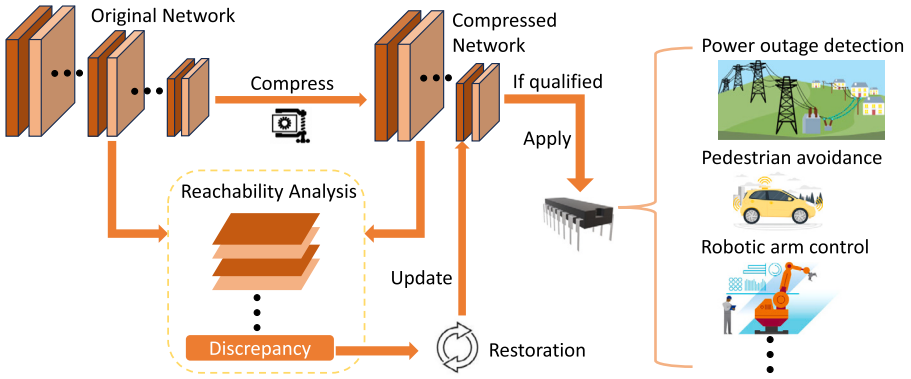


**Fig. 6.** Prospects for engineering applications based on reachability-based discrepancy computation.

## 5. Engineering applications

Our proposed maximum output discrepancy computation method provides an exact value to characterize the output discrepancy between the original network and the compressed network with the given input set. Here, our work not only provides a reachability-based discrepancy computation method but also a new metric to evaluate the performance of compression methods. The discrepancy values can reveal some critical properties in extreme scenarios, such as vulnerability against adversarial attacks, where accuracy metrics cannot represent. As shown in experiment Section 4.2, even though the compressed network has the same accuracy as the original one, the discrepancy is significantly different, which implies that the vulnerability is significantly different between the two networks in certain extreme scenarios, it may cause critical safety hazards in safety-sensitive engineering applications. For a compressed network, a lower discrepancy means a better retention of the original network.

In engineering applications, large systems are often composed of multiple small systems; among them, edge devices are assigned many different tasks. Nowadays, many edge devices are deployed with small neural networks to improve the large system's efficiency, flexibility, and robustness [4,33]. However, due to the hardware limitations of edge devices, those deployed networks are required to be small in scale and have a low energy cost. Thus, compressed networks are the primary solution for these edge devices. As stated in Section 2, the compression process may cause a significant drop-down in performance, which may deteriorate the sensitive system and result in severe consequences. As many compression methods research only recognize accuracy as the criterion for evaluation, they are not safe enough for directly deploying to safety-sensitive edge devices. For example, edge devices are used in power outage detection [12,5]. In the power grid, to provide early warning of possible power outages, such detection devices need to run continuously for a long time and be independent of the power grid. This requires that the compressed network on the equipment can run for a long time at a low cost and is robust enough to distinguish false positive warning information. For the pedestrian recognition and avoidance situation [22,43], the processing system requires real-time and rapid analysis capability, where a small, fast, and robust compressed network is competent for the job. Otherwise, unexpected accidents may happen due to the lagging results. Besides, in large-scale digital industries, many robotic arms must complete their respective tasks accurately without error [8,18]. To improve the efficiency and benefits, robotic arms should be controlled at a low cost and robust to perturbation, where the high accuracy of compressed networks can't guarantee safety robustness. More and more engineering applications require not just a tiny and fast network but a robust, reliable, and flexible network to accomplish cumbersome tasks, where our method provides an approach to verify the robustness of the compressed network at a practical level.

In Fig. 6, we show our prospects for engineering applications based on our discrepancy computation method. Before deploying the compressed network, we use our method to compute the discrepancy with the original network. If the discrepancy meets the prescribed criteria, the compressed network can apply to edge devices. If the discrepancy is out of permissible range, the restoration process, which is our next research topic, can restore the compressed network to the original network and update it. The loop for restoration will end when the compressed network passes the discrepancy evaluation. After this whole analysis and restoration process, the compressed network is safe and robust enough to be deployed to those engineering applications.

## 6. Conclusions and further remarks

This work proposes an approach to compute the difference between two convolutional neural networks by giving the error range for the output difference. It formally defines constructing the merged neural network from two given convolutional neural networks. Via the merging approach, an ImageStar-based computation procedure is developed to compute the reachable set of the difference between two networks' output with the same input. Then, our approach successfully gives quantitative analysis results between the original and quantization networks by giving the error range output, which represents the difference in each dimension. The task is carried out by applying the error range to the original network's output to verify the quantization method, as shown by the Cifar10 example.

For future work, we aim to optimize the time complexity of our method and increase its efficiency with more extensive network computations. We plan to implement parallel computing in our method to speed up reachable set computation and fully use the GPU

power in matrix computation. Besides, we plan to support more layer operations, such as batch normalization, dropout, etc., and give more accurate overapproximation for other non-linear activation functions, such as Softmax, Sigmoid, leaky-ReLU, etc. With the development of the methods, we can implement and compare more different compression methods to identify which one has better performance on minor discrepancies with the original network. Furthermore, we aim to use the analysis results best to restore the compression network by applying the discrepancy during the training process to narrow down further the difference between the compression and original networks.

## CRediT authorship contribution statement

**Zihao Mo:** Investigation, Methodology, Software, Validation, Writing – original draft. **Weiming Xiang:** Conceptualization, Methodology, Supervision, Writing – review & editing.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Weiming Xiang reports financial support was provided by National Science Foundation, under NSF CAREER Award no. 2143351, NSF CNS Award no. 2223035, and NSF IIS Award 2331938.

## Data availability

The data and code are publicly available in the shared link in the paper.

## References

[1] Lei Deng, Guoqi Li, Song Han, Luping Shi, Yuan Xie, Model compression and hardware acceleration for neural networks: a comprehensive survey, Proc. IEEE 108 (4) (2020) 485–532.
[2] Claudio Ferrari, Mark Niklas Müller, Nikola Jovanovic, Martin T. Vechev, Complete verification via multi-neuron relaxation guided branch-and-bound, in: The Tenth International Conference on Learning Representations, ICLR 2022 Poster, Virtual Event, April 25-29, 2022, OpenReview.net, 2022.
[3] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, Explaining and harnessing adversarial examples, arXiv preprint, arXiv:1412.6572, 2014.
[4] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, William J. Dally Eie, Efficient inference engine on compressed deep neural network, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 243–254.
[5] Carmen Haseltine, Eman El-Sheikh Eman, Prediction of power grid failure using neural network learning, in: 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2017, pp. 505–510.
[6] Patrick Henriksen, Alessio Lomuscio, Efficient neural network verification via adaptive refinement and adversarial search, in: European Conference on Artificial Intelligence, ECAI 2020, IOS Press, 2020, pp. 2513–2520.
[7] Patrick Henriksen, Alessio Lomuscio Deepsplit, An efficient splitting method for neural network verification via indirect effect analysis, in: International Joint Conferences on Artificial Intelligence, IJCAI, 2021, pp. 2549–2555.
[8] Ted Hesselroth, Kakali Sarkar, P. Patrick Van der Smagt, Klaus Schulten, Neural network control of a pneumatic robot arm, IEEE Trans. Syst. Man Cybern. 24 (1) (1994) 28–38.
[9] Geoffrey Hinton, Oriol Vinyals, Jeffrey Dean, Distilling the knowledge in a neural network, in: NIPS Deep Learning and Representation Learning Workshop, 2015.
[10] Zhifu Jia, Cunlin Li, Almost sure exponential stability of uncertain stochastic Hopfield neural networks based on subadditive measures, Mathematics 11 (14) (2023).
[11] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, Mohammed Bennamoun, A guide to convolutional neural networks for computer vision, Synth. Lect. Comput. Vis. 8 (1) (2018) 1–207.
[12] Sifat Shahriar Khan, Jin Wei, Real-time power outage detection system using social sensing and neural networks, in: 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP), IEEE, 2018, pp. 927–931.
[13] Fanchao Kong, Quanxin Zhu, Fixed-time stabilization of discontinuous neutral neural networks with proportional delays via new fixed-time stability lemmas, IEEE Trans. Neural Netw. Learn. Syst. 34 (2) (2023) 775–785.
[14] Raghuraman Krishnamoorthi, Quantizing deep convolutional networks for efficient inference: a whitepaper, arXiv preprint, arXiv:1806.08342, 2018.
[15] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, CIFAR-10 (Canadian Institute for Advanced Research), Dataset, 2009.
[16] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012.
[17] S. Lawrence, C.L. Giles, Ah Chung Tsoi, A.D. Back, Face recognition: a convolutional neural-network approach, IEEE Trans. Neural Netw. 8 (1) (1997) 98–113.
[18] Shuai Li, Yinyan Zhang, Neural Networks for Cooperative Control of Multiple Robot Arms, Springer, 2018.
[19] Zhen Liang, Dejin Ren, Wanwei Liu, Ji Wang, Wenjing Yang, Bai Xue, Safety verification for neural networks based on set-boundary analysis, in: International Symposium on Theoretical Aspects of Software Engineering, Springer, 2023, pp. 248–267.
[20] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, Clara I. Sánchez, A survey on deep learning in medical image analysis, Med. Image Anal. 42 (2017) 60–88.
[21] Pierre-Jean Meyer, Reachability analysis of neural networks using mixed monotonicity, IEEE Control Syst. Lett. 6 (2022) 3068–3073.
[22] Eike Rehder, Florian Wirth, Martin Lauer, Christoph Stiller, Pedestrian prediction by planning using deep neural networks, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 5903–5908.
[23] Jürgen Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[24] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, Martin Vechev, Fast and effective robustness certification, Adv. Neural Inf. Process. Syst. 31 (2018) 10802–10813.
[25] Gagandeep Singh, Timon Gehr, Markus Püschel, Martin Vechev, An abstract domain for certifying neural networks, Proc. ACM Program. Lang. 3 (POPL) (2019) 1–30.
[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus, Intriguing properties of neural networks, arXiv e-prints, arXiv:1312.6199, December 2013.

[27] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, Gabriel F. Manso, The computational limits of deep learning, arXiv preprint, arXiv:2007.05558, 2020.
[28] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, Taylor T. Johnson, Verification of deep convolutional neural networks using imagestars, in: Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I, Springer, 2020, pp. 18–42.
[29] Hoang Dung Tran, Sung Woo Choi, Xiaodong Yang, Tomoya Yamaguchi, Bardh Hoxha, Danil Prokhorov, Verification of recurrent neural networks with star reachability, in: Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, 2023, pp. 1–13.
[30] Hoang-Dung Tran, Sungwoo Choi, Hideki Okamoto, Bardh Hoxha, Georgios Fainekos, Danil Prokhorov, Quantitative verification for neural networks using probstars, in: Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, 2023, pp. 1–12.
[31] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, Taylor T. Johnson, Star-based reachability analysis of deep neural networks, in: Formal Methods–The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings 3, Springer, 2019, pp. 670–686.
[32] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, Taylor T. Johnson Nnv, The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems, in: Shuvendu K. Lahiri, Chao Wang (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2020, pp. 3–17.
[33] Ya Tu, Yun Lin, Deep neural network compression technique towards efficient digital signal modulation recognition in edge device, IEEE Access 7 (2019) 58113–58119.
[34] Lijun Wang, Wanli Ouyang, Xiaogang Wang, Huchuan Lu, Visual tracking with fully convolutional networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 3119–3127.
[35] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, J. Zico Kolter, Beta-crown: efficient bound propagation with per-neuron split constraints for neural network robustness verification, in: M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, J. Wortman Vaughan (Eds.), Advances in Neural Information Processing Systems, vol. 34, Curran Associates, Inc., 2021, pp. 29909–29921.
[36] Simon Wiedemann, Heiner Kirchhoffer, Stefan Matlage, Paul Haase, Arturo Marban, Talmaj Marinč, David Neumann, Tung Nguyen, Heiko Schwarz, Thomas Wiegand, Detlev Marpe, Wojciech Samek, Deepcabac: a universal compression algorithm for deep neural networks, IEEE J. Sel. Top. Signal Process. 14 (4) (2020) 700–714.
[37] Weiming Xiang, Zhongzhu Shao, Approximate bisimulation relations for neural networks and application to assured neural network compression, in: 2022 American Control Conference (ACC), IEEE, 2022, pp. 3248–3253.
[38] Weiming Xiang, Hoang-Dung Tran, Taylor T. Johnson, Output reachable set estimation and verification for multilayer neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (11) (2018) 5777–5783.
[39] Weiming Xiang, Hoang-Dung Tran, Joel A. Rosenfeld, Taylor T. Johnson, Reachable set estimation and safety verification for piecewise linear systems with neural network controllers, in: 2018 Annual American Control Conference (ACC), IEEE, 2018, pp. 1574–1579.
[40] Weiming Xiang, Hoang-Dung Tran, Xiaodong Yang, Taylor T. Johnson, Reachable set estimation for neural network control systems: a simulation-guided approach, IEEE Trans. Neural Netw. Learn. Syst. 32 (5) (2021) 1821–1830.
[41] Tien-Ju Yang, Yu-Hsin Chen, Vivienne Sze, Designing energy-efficient convolutional neural networks using energy-aware pruning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5687–5695.
[42] Yejiang Yang, Tao Wang, Jefferson P. Woolard, Weiming Xiang, Guaranteed approximation error estimation of neural networks and model modification, Neural Netw. 151 (2022) 61–69.
[43] Shuai Yi, Hongsheng Li, Xiaogang Wang, Pedestrian behavior understanding and prediction with deep neural networks, in: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14, Springer, 2016, pp. 263–279.
[44] Chi Zhang, Wenjie Ruan, Fu Wang, Peipei Xu, Geyong Min, Xiaowei Huang, Model-agnostic reachability analysis on deep neural networks, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2023, pp. 341–354.
[45] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, J. Zico Kolter, General cutting planes for bound-propagation-based neural network verification, in: by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, vol. 35, Curran Associates, Inc., 2022, pp. 1656–1670.
[46] Yabo Zhang, Wenrui Ding, Chunlei Liu, Summary of convolutional neural network compression technology, in: 2019 IEEE International Conference on Unmanned Systems (ICUS), 2019, pp. 480–483.
[47] Yuhao Zhang, Xiangru Xu, Reachability analysis and safety verification of neural feedback systems via hybrid zonotopes, in: 2023 American Control Conference (ACC), IEEE, 2023, pp. 1915–1921.
[48] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, Yurong Chen, Incremental network quantization: towards lossless CNNs with low-precision weights, in: International Conference on Learning Representations, ICLR Poster, 2017.
[49] Song Zhu, Yu Gao, Yuxin Hou, Chunyu Yang, Reachable set estimation for memristive complex-valued neural networks with disturbances, IEEE Trans. Neural Netw. Learn. Syst. (2022) 1–6, Early Access.