Verification of Deep Convolutional Neural Networks using ImageStars *

Hoang-Dung Tran[1,2], Stanley Bak[3], Weiming Xiang[4], and Taylor T. Johnson[2]

[1] University of Nebraska, USA
[2] Vanderbilt University, USA
[3] Stony Brook University, USA
[4] Augusta University, USA

Abstract. Convolutional Neural Networks (CNN) have redefined state-of-the-art in many real-world applications, such as facial recognition, image classification, human pose estimation, and semantic segmentation. Despite their success, CNNs are vulnerable to adversarial attacks, where slight changes to their inputs may lead to sharp changes in their output in even well-trained networks. Set-based analysis methods can detect or prove the absence of bounded adversarial attacks, which can then be used to evaluate the effectiveness of neural network training methodology. Unfortunately, existing verification approaches have limited scalability in terms of the size of networks that can be analyzed. In this paper, we describe a set-based framework that successfully deals with real-world CNNs, such as VGG16 and VGG19, that have high accuracy on ImageNet. Our approach is based on a new set representation called the ImageStar, which enables efficient exact and over-approximative analysis of CNNs. ImageStars perform efficient set-based analysis by combining operations on concrete images with linear programming (LP). Our approach is implemented in a tool called NNV, and can verify the robustness of VGG networks with respect to a small set of input states, derived from adversarial attacks, such as the DeepFool attack. The experimental results show that our approach is less conservative and faster than existing zonotope and polytope methods.

Keywords: neural networks · reachability analysis · machine learning · computer vision

* The material presented in this paper is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089, the National Science Foundation (NSF) under grant numbers SHF 1910017 and FMitF 1918450, and the Air Force Office of Scientific Research (AFOSR) through award numbers FA9550-18-1-0122 and FA9550-19-1-0288. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR, DARPA, or NSF. An extended version of this paper with appendices is available [33].

1 Introduction

Convolutional neural networks (CNN) have rapidly accelerated progress in computer vision with many practical applications such as face recognition [19], image classification [18], document analysis [21] and semantic segmentation. Recently, it has been shown that CNNs are vulnerable to adversarial attacks, where a well-trained CNN can be fooled into producing errant predictions due to tiny changes in their inputs [9]. Many applications such as autonomous driving seek to leverage the power of CNNs. However due the opaque nature of these models there are reservations about using in safety-critical applications. Thus, there is an urgent need for formally evaluating the robustness of a trained CNN.

Formal verification of deep neural networks (DNNs) has recently become an important topic. The majority of existing approaches focus on verifying safety and robustness properties of feedforward neural networks (FNN) with the Rectified Linear Unit activation function (ReLU). These approaches include: mixed-integer linear programming (MILP) [5, 17, 23], satisfiability (SAT) and satisfiability modulo theory (SMT) techniques [7, 15], optimization [6, 11, 22, 42, 44, 51], and geometric reachability [29, 30, 36, 37, 41, 43, 45, 47, 48, 50]. Adjacent to these methods are property inference techniques for DNNs, which are also an important and interesting research area being investigated [10]. In a similar fashion, the problem of verifying safety of cyber-physical systems (CPS) with learning-enabled neural network components with imperfect plant models and sensing information has recently attracted significant attention due to their real world applications [1,12–14,24,31,32,35,46,49]. This research area views the safety verification problem in a holistic manner by considering safety of the entire system where learning-enabled components interact with the physical world.

Although numerous tools and methods have been proposed for neural network verification, only a handful of methods can deal with CNNs [2, 16, 17, 27, 29, 30]. Moreover, in the aforementioned techniques, only one [27] can deal with real-world CNNs, such as VGGNet [28]. Their approach makes used of the concept of the L_0 distance between two images. Their optimization-based approach computes a tight bound on the number of pixels that may be changed in an image without affecting the classification result of the network. It can also efficiently generate adversarial examples that can be used to improve the robustness of network. In a similar manner, this paper seeks to verify robustness of real-world deep CNNs. Thus, we develop a set-based analysis method through the use of the *ImageStar*, a new set representation that can represent an infinite family of images. As an example, this representation can be used to represent a set of images distorted by an adversarial attack. Using the ImageStar, we develop both exact and over-approximate reachability algorithms to construct reachable sets that contain all the possible outputs of a CNN under an adversarial attack. These reachable sets are then used to reason about the overall robustness of the network. When a CNN violates a robustness property, our exact reachability scheme can construct a *set of concrete adversarial examples*. Our approach differs from [27] in two primary ways. First, our method does not provide robustness guarantees for a network in terms of the number of pixels that are allowed

to be changed (in terms of L_0 distance). Instead, we prove the robustness of the network on images that are attacked by disturbances bounded by arbitrary linear constraints. Second, our approach relies on reachable set computation of a network corresponding to a bounded input set, as opposed to an optimization-based approach. We implement these methods in the NNV tool [39] and compare with the zonotope method used in DeepZ [29] and the polytope method used in DeepPoly [30]. The experimental results indicate our method is less conservative and faster than existing approaches when verifying robustness of CNNs.

The main contributions of the paper include the following. First is the ImageStar set representation, which is an efficient representation for reachability analysis of CNNs. Second are exact and over-approximate reachability algorithms for constructing reachable sets and verifying robustness of CNNs. Third is the implementation of the ImageStar representation and reachability algorithms in NNV [39]. Fourth is a rigorous evaluation and comparison of proposed approaches, such as zonotope and polytope methods on different CNNs.

2 Problem formulation

The reachability problem for CNNs is the task of analyzing a trained CNN with respect to some perturbed input set in order to construct a set containing all possible outputs of the network. In this paper, we consider the reachability of a CNN $\mathcal{N}$ that consists of a series of layers L that may include convolutional layers, fully connected layers, max-pooling layers, average pooling layers, and ReLU activation layers. Mathematically, we define a CNN with n layers as $\mathcal{N} = \{L_i\}, i = 1, 2, \ldots, n$. The reachability of the CNN $\mathcal{N}$ is defined based on the concept of *reachable sets*.

Definition 1 (Reachable set of a CNN). *An (output) reachable set $\mathcal{R}_\mathcal{N}$ of a CNN $\mathcal{N} = \{L_i\}, i = 1, 2, \ldots, n$ corresponding to a linear input set $\mathcal{I}$ is defined incrementally as:*

$$\begin{aligned}
\mathcal{R}_{L_1} &\triangleq \{y_1 \mid y_1 = L_1(x),\ x \in \mathcal{I}\},\\
\mathcal{R}_{L_2} &\triangleq \{y_2 \mid y_2 = L_2(y_1),\ y_1 \in \mathcal{R}_{L_1}\},\\
&\vdots\\
\mathcal{R}_\mathcal{N} = \mathcal{R}_{L_n} &\triangleq \{y_n \mid y_n = L_n(y_{n-1}),\ y_{n-1} \in \mathcal{R}_{L_{n-1}}\},
\end{aligned}$$

where $L_i(\cdot)$ is a function representing the operation of the i^{th} layer.

The definition shows that the reachable set of the CNN $\mathcal{N}$ can be constructed *layer-by-layer*. The core computation is constructing the reachable set of each layer L_i defined by a specific operation, i.e., convolution, affine mapping, max pooling, average pooling, or ReLU.

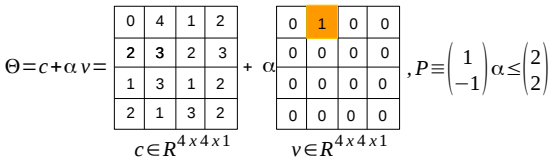

Fig. 1: An example of an ImageStar.

3 ImageStar

Definition 2. ***An ImageStar*** Θ *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,* $P : \mathbb{R}^m \to \{\top, \bot\}$ *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:*

$$[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m}(\alpha_i v_i) \text{ such that } P(\alpha_1, \cdots, \alpha_m) = \top\}.$$

Sometimes we will refer to both the tuple Θ *and the set of states* $[\![\Theta]\!]$ *as* Θ*. 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,* $C \in \mathbb{R}^{p\times m}$*,* α *is the vector of* m*-variables, i.e.,* $\alpha = [\alpha_1, \cdots, \alpha_m]^T$*, and* $d \in \mathbb{R}^{p\times 1}$*. A ImageStar is an empty set if and only if* $P(\alpha)$ *is empty.*

Example 1 (ImageStar). A $4 \times 4 \times 1$ gray image with a bounded disturbance $b \in [-2, 2]$ applied on the pixel of the position $(1, 2, 1)$ can be described as an ImageStar depicted in Figure 1.

Remark 1. An ImageStar is an extension of the generalized star set recently defined in [3, 4, 37, 38]. In a generalized star set, the anchor and the generators are vectors, while in an ImageStar, the anchor and generators are images with multiple channels. We will later show that the ImageStar is a very efficient representation for the reachability analysis of convolutional layers, fully connected layers, and average pooling layers.

Proposition 1 (Affine mapping of an ImageStar). *An affine mapping of an ImageStar* $\Theta = \langle c, V, P\rangle$ *with a scale factor* γ *and an offset image* β *is another ImageStar* $\Theta' = \langle c', V', P'\rangle$ *in which the new anchor, generators and predicate are as follows:*

$$c' = \gamma \times c + \beta, \quad V' = \gamma \times V, \quad P' \equiv P.$$

Note that, the scale factor γ *can be a scalar or a vector containing scalar scale factors in which each factor is used to scale one channel in the ImageStar.*

4 Reachability of CNN using ImageStars

In this section, we present the reachable set computation for the convolutional, average pooling, fully connected, batch normalization, max pooling, and ReLU layers with respect to an input set consisting of an ImageStar.

4.1 Reachability of a convolutional layer

We consider a two-dimensional convolutional layer with following parameters: the weights $W_{Conv2d} \in \mathbb{R}^{h_f \times w_f \times nc \times nf}$, the bias $b_{Conv2d} \in \mathbb{R}^{1\times 1\times nf}$, the padding size P, the stride S, and the dilation factor D where h_f, w_f, nc are the height, width, and the number of channels of the filters in the layer respectively. Additionally, nf is the number of filters. The reachability of a convolutional layer is given in the following lemma.

Lemma 1. *The reachable set of a convolutional layer with an ImageStar input set $\mathcal{I} = \langle c, V, P\rangle$ is another ImageStar $\mathcal{I}' = \langle c', V', P\rangle$ where $c' = Convol(c)$ is the convolution operation applied to the anchor image, $V' = \{v'_1, \ldots, v'_m\}, v'_i = ConvolZeroBias(v_i)$ is the convolution operation with zero bias applied to the generator images, i.e., only using the weights of the layer.*

Proof. Any image in the ImageStar input set is a *linear* combination of the center and basis images. For any filter in the layer, the convolution operation applied to the input image performs local element-wise multiplication of a local matrix (of all channels) containing the values of the local pixels of the image and the the weights of the filter and then combine the result with the bias to get the output for that local region. Due to the linearity of the input image, we can perform the convolution operation with the bias on the center and the convolution operation with zero bias on the basis images and then combine the result to get the output image.

Example 2 (Reachable set of a convolutional layer). The reachable set of a convolutional layer with single 2×2 filter and the ImageStar input set in Example 1 is described in Figure 2, where the weights and the bias of the filter are $W = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}$, $b = -1$ respectively, the stride is $S = [2\ 2]$, the padding size is $P = [0\ 0\ 0\ 0]$ and the dilation factor is $D = [1\ 1]$.

4.2 Reachability of an average pooling layer

The reachability of an average pooling layer with pooling size PS, padding size P, and stride S is given below, with its proof similar to that of the convolutional layer.

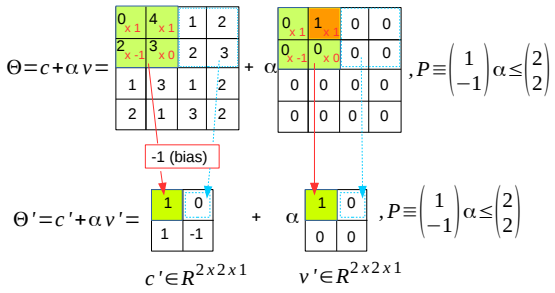

Fig. 2: Reachability of convolutional layer using ImageStar.

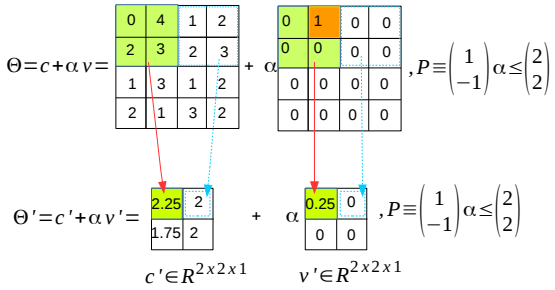

Fig. 3: Reachability of average pooling layer using ImageStar.

Lemma 2. *The reachable set of a average pooling layer with an ImageStar input set* $\mathcal{I} = \langle c, V, P \rangle$ *is another ImageStar* $\mathcal{I}' = \langle c', V', P \rangle$ *where* $c' = average(c)$, $V' = \{v'_1, \ldots, v'_m\}, v'_i = average(v_i)$, $average(\cdot)$ *is the average pooling operation applied to the anchor and generator images.*

Example 3 (Reachable set of an average pooling layer). The reachable set of an 2×2 average pooling layer with padding size $P = [0\ 0\ 0\ 0]$, stride $S = [2\ 2]$, and an ImageStar input set given by Example 1 is shown in Figure 3.

4.3 Reachability of a fully connected layer

The reachability of a fully connected layer is stated in the following lemma.

Lemma 3. *Given a two-dimensional fully connected layer with weight* $W_{fc} \in \mathbb{R}^{n_{fc} \times m_{fc}}$, *bias* $b_{fc} \in \mathbb{R}^{n_{fc}}$, *and an ImageStar input set* $\mathcal{I} = \langle c, V, P \rangle$, *the reachable set of the layer is another ImageStar* $\mathcal{I}' = \langle c', V', P \rangle$ *where* $c' = W * \bar{c} + b$, $V' = \{v'_1, \ldots, v'_m\}, v'_i = W_{fc} * \bar{v}_i$, $\bar{c}(\bar{v}_i) = reshape(c(v_i), [m_{fc}, 1])$. *Note that it is required for consistency between the ImageStar and the weight matrix that* $m_{fc} = h \times w \times nc$, *where* h, w, nc *are the height, width and number of channels of the ImageStar.*

Proof. Similar to the convolutional layer and the average pooling layer, for any image in the ImageStar input set, the fully connected layer performs an affine

mapping of the input image which is a linear combination of the center and the basis images of the ImageStar. Due to the linearity, the affine mapping of the input image can be decomposed into the affine mapping of the center image and the affine mapping without the bias of the basis images. The final result is the sum of the individual affine maps.

4.4 Reachability of a batch normalization layer

In the prediction phase, a batch normalization layer normalizes each input channel x_i using the mean μ and variance σ^2 over the full training set. Then the batch normalization layer further shifts and scales the activations using the offset β and the scale factor γ that are learnable parameters. The formula for normalization is as follows:

$$\bar{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y_i = \gamma \bar{x}_i + \beta,$$

where ϵ is a used to prevent division by zero. The batch normalization layer can be described as a tuple $\mathcal{B} = \langle \mu, \sigma^2, \epsilon, \gamma, \beta \rangle$. The reachability of a batch normalization layer with an ImageStar input set is given in the following lemma.

Lemma 4. *The reachable set of a batch normalization layer* $\mathcal{B} = \langle \mu, \sigma^2, \epsilon, \gamma, \beta \rangle$ *with an ImageStar input set* $\mathcal{I} = \langle c, V, P \rangle$ *is another ImageStar* $\mathcal{I}' = \langle c', V', P' \rangle$ *where:*

$$c' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} c + \beta - \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \mu, \quad V' = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} V, \quad P' \equiv P.$$

The reachable set of a batch normalization layer can be obtained in a straightforward fashion using two affine mappings of the ImageStar input set.

4.5 Reachability of a max pooling layer

Reachability of max pooling layer with an ImageStar input set is challenging because the value of each pixel in an image in the ImageStar depends on the predicate variables α_i. Therefore, the local max point when applying max-pooling operation may change with the values of the predicate variables. In this section, we investigate the exact reachability and over-approximate reachability of a max pooling layer with an ImageStar input set. The first obtains the exact reachable set while the second constructs an over-approximate reachable set.

Exact reachability of a max pooling layer The central idea in the exact analysis of the max-pooling layer is finding a set of *local max point candidates* when we apply the max pooling operation on the image. We consider the max pooling operation on the ImageStar in Example 1 with a pool size of 2×2, a padding size of $P = [0\ 0\ 0\ 0]$, and a stride $S = [2\ 2]$ to clarify the exact analysis step-by-step. First, the max-pooling operation is applied on 4 local

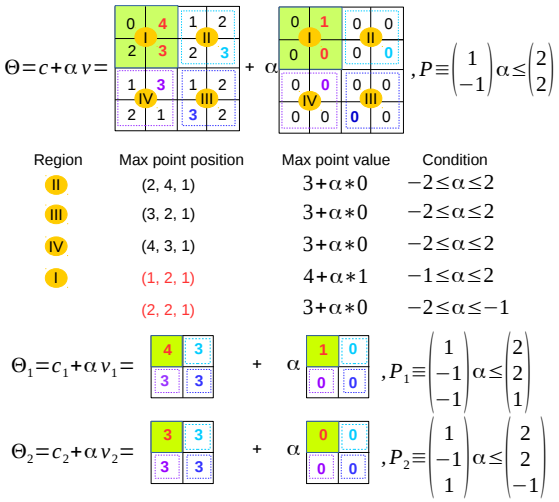

Fig. 4: Exact reachability of max pooling layer using ImageStars.

regions I, II, III, IV, as shown in Figure 4. The local regions II, III, IV have only one *max point candidate* whic is the pixel that has the maximum value in the region. It is interesting to note that region I has two max point candidates at the positions $(1, 2, 1)$ and $(2, 2, 1)$ and these candidates correspond to different conditions of the predicate variable α. For example, the pixel at the position $(1, 2, 1)$ is the max point if and only if $4 + \alpha \times 1 \geq 3 + \alpha \times 0$. Note that with $-2 \leq \alpha \leq 2$, we always have $4 + \alpha * 1 \geq 2 + \alpha \times 0 \geq 0 + \alpha \times 0$. Since the local region I has two max point candidates, and other regions have only one, the exact reachable set of the max-pooling layer is the union of two new ImageStars Θ_1 and Θ_2. In the first reachable set Θ_1, the max point of the region I is $(1, 2, 1)$ with an additional constraint on the predicate variable $\alpha \geq -1$. For the second reachable set Θ_2, the max point of the region I is $(2, 2, 1)$ with an additional constraint on the predicate variable $\alpha \leq -1$. One can see that from a single ImageStar input set, the output reachable set of the max-pooling layer is split into two new ImageStars. Therefore, the number of ImageStars in the reachable set of the max-pooling layer may grow quickly if each local region has more than one max point candidates. The worst-case complexity of the number of ImageStars in the exact reachable set of the max-pooling layer is given in Lemma 5. The exact reachability algorithm is presented in the Appendix A.1, Algorithm 2.

Lemma 5. *The worst-case complexity of the number of ImageStars in the exact reachability of the max pooling layer is* $\mathcal{O}(((p_1 \times p_2)^{h \times w})^{nc})$ *where* $[h, w, nc]$ *is the size of the ImageStar output sets, and* $[p_1, p_2]$ *is the size of the max-pooling layer.*

Proof. An image in the ImageStar output set has $h \times w$ pixels in each channel. For each pixel, in the worst case, there are $p_1 \times p_2$ candidates. Therefore, the number of ImageStars in the output set in the worst case is $\mathcal{O}(((p_1 \times p_2)^{h \times w})^{nc})$.

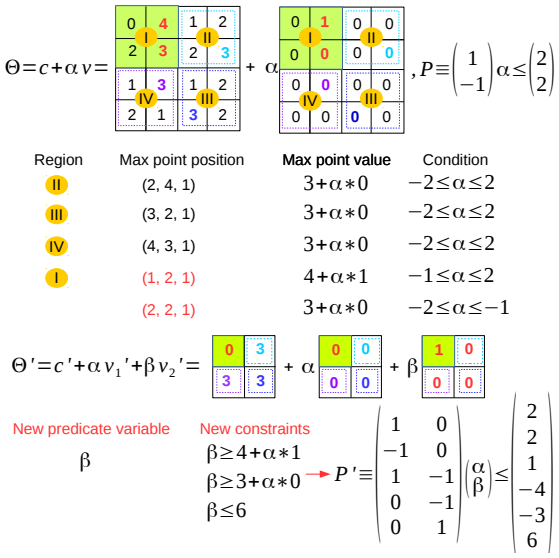

Fig. 5: Over-approximate reachability of max pooling layer using ImageStar.

Finding a set of local max point candidates is the core computation in the exact reachability of max-pooling layer. To optimize this computation, we divide the search for the local max point candidates into two steps. The first one is to estimate the ranges of all pixels in the ImageStar input set. We can solve $h_I \times w_I \times nc$ linear programming optimizations to find the exact ranges of these pixels, where $[h_I, w_I, nc]$ is the size of the input set. However, unfortunately this is a time-consuming computation. For example, ***if a single linear optimization can be done in*** 0.01 ***seconds, for an ImageStar of the size*** $224 \times 224 \times 32$***, we need about*** 10 ***hours to find the ranges of all pixels***. To overcome this bottleneck, we quickly estimate the ranges using only the ranges of the predicate variables to get rid of a vast amount of non-max-point candidates. In the second step, we solve a much smaller number of LP optimizations to determine the exact set of the local max point candidates and then construct the ImageStar output set based on these candidates.

Lemma 5 shows that the number of ImageStars in the exact reachability analysis of a max-pooling layer may grow exponentially. To overcome this problem, we propose the following over-approximate reachability method.

Over-approximate reachability of a max pooling layer The central idea of the over-approximate analysis of the max-pooling layer is that if a local region has more than one max point candidates, we introduce a *new predicate variable* standing for the max point of that region. We revisit the example introduced earlier in the exact analysis to clarify this idea. Since the first local region I has two max point candidates, we introduce new predicate variable β to represent the max point of this region by adding three new constraints: 1) $\beta \geq 4+\alpha*1$, i.e., β must be equal or larger than the value of the first candidate ; 2) $\beta \geq 3+\alpha*0$,

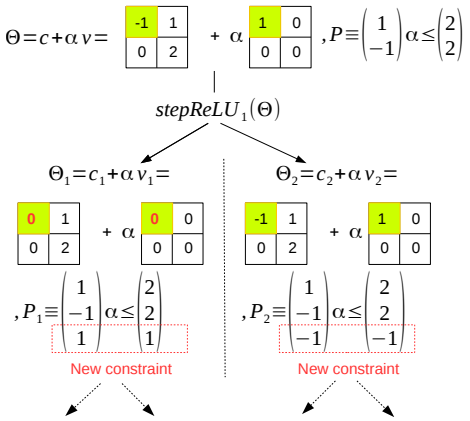

Fig. 6: stepReLU operation on an ImageStar.

i.e., β must be equal or larger than the value of the second candidate; 3) $\beta \leq 6$, i.e., β must be equal or smaller than the upper bound of the pixels values in the region. With the new predicate variable, a single over-approximate reachable set Θ' can be constructed in Figure 5. The approximate reachability algorithm is presented in the Appendix A.2, Algorithm 3.

Lemma 6. *The worst-case complexity of the new predicate variables introduced in the over-approximate analysis is* $\mathcal{O}(h \times w \times nc)$ *where* $[h, w, nc]$ *is the size of the ImageStar output set.*

4.6 Reachability of a ReLU layer

Similar to max-pooling layer, the reachability analysis of a ReLU layer is also challenging because the value of each pixel in an ImageStar may be smaller than zero or larger than zero depending on the values of the predicate variables ($ReLU(x) = max(0, x)$). In this section, we investigate the exact and over-approximate reachability algorithms for a ReLU layer with an ImageStar input set. The techniques we use in this section are adapted from in [37].

Exact reachability of a ReLU layer The central idea of the exact analysis of a ReLU layer with an ImageStar input set is performing a sequence of *stepReLU operations* over all pixels of the ImageStar input set. Mathematically, the exact reachable set of a ReLU layer L can be computed as follows.

$$\mathcal{R}_L = stepReLU_N(stepReLU_{N-1}(\dots(stepReLU_1(\mathcal{I})))),$$

where N is the total number of pixels in the ImageStar input set $\mathcal{I}$. The $stepReLU_i$ operation determines whether or not a split occurs at the i^{th} pixel. If the pixel value is larger than zero, then the output value of that pixel remains the same. If the pixel value is smaller than zero than the output value of that pixel is reset to be zero. The challenge is that the pixel value depends on the predicate variables.

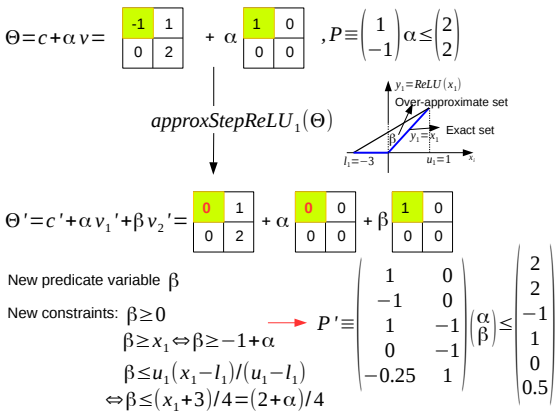

Fig. 7: approxStepReLU operation on an ImageStar.

Therefore, there is the case that the pixel value may be negative or positive with *an extra condition* on the predicate variables. In this case, we split the input set into two *intermediate* ImageStar reachable sets and apply the ReLU law on each intermediate reach set. An example of the stepReLU operation on an ImageStar is illustrated in Figure 6. The value of the first pixel value $-1 + \alpha$ would be larger than zero if $\alpha \leq 1$, and in this case we have $ReLU(-1 + \alpha) = -1 + \alpha$. If $\alpha <= 1$, then $ReLU(-1 + \alpha) = 0 + \alpha \times 0$. Therefore, the first stepReLU operation produces two intermediate reachable sets Θ_1 and Θ_2, as shown in the figure. The number of ImageStars in the exact reachable set of a ReLU layer increases quickly along with the number of splits in the analysis, as stated in the following lemma.

Lemma 7. *The worst-case complexity of the number of ImageStars in the exact analysis of a ReLU layer is* $\mathcal{O}(2^N)$*, where* N *is the number of pixels in the ImageStar input set.*

Proof. There are $h \times w \times nc$ local regions in the approximate analysis. In the worst case, we need to introduce a new variable for each region. Therefore, the worst case complexity of new predicate variables introduced is $\mathcal{O}(h \times w \times nc)$.

Similar to [37], to control the explosion in the number of ImageStars in the exact reachable set of a ReLU layer, we propose an over-approximate reachability algorithm in the following.

Over-approximate reachability of a ReLU layer The idea behind the over-approximate reachability of ReLU layer is replacing the stepReLU operation at each pixel in the ImageStar input set by an *approxStepReLU* operation. At each pixel where a split occurs, we introduce a new predicate variable to over-approximate the result of the stepReLU operation at that pixel. An example of the overStepReLU operation on an ImageStar is depicted in Figure 7 in which the first pixel of the input set has the ranges of $[l_1 = -3, u_1 = 1]$ indicating that

Algorithm 1 Reachability analysis for a CNN.

Input: $\mathcal{N} = \{L_i\}_1^n$, $\mathcal{I}$, *scheme* (`'exact' or 'approx'`)
Output: $R_\mathcal{N}$
1: **procedure** $R_\mathcal{N}$ = REACH($\mathcal{N}, \mathcal{I}, scheme$)
2: $In = \mathcal{I}$
3: **parfor** $i = 1 : n$ **do** $In = L_i.reach(In, scheme)$
4: **end parfor**
5: $R_\mathcal{N} = In$

a split occurs at this pixel. To avoid this split, we introduce a new predicate variable β to over-approximate the exact intermediate reachable set (i.e., two blue segments in the figure) by a triangle. This triangle is determined by three constraints: 1) $\beta \geq 0$ (the $ReLU(x) \geq 0$ for any x); 2) $\beta \geq -1+\alpha$ ($ReLU(x) \geq x$ for any x); 3) $\beta \leq 0.5+0.25\alpha$ (upper bound of the new predicate variable). Using this over-approximation, a single intermediate reachable set Θ' is produced as shown in the figure. After performing a sequence of approxStepReLU operations, we obtain a single over-approximate ImageStar reachable set for the ReLU layer. However, the number of predicate variables and the number of constraints in the obtained reachable set increase.

Lemma 8. *The worst case complexity of the increment of predicate variables and constraints is $\mathcal{O}(N)$ and $\mathcal{O}(3 \times N)$ respectively, where N is the number of pixels in the ImageStar input set.*

Proof. In the worst case, splits occur at all N pixels in the ImageStar input set. In this case, we need to introduce N new predicate variables to over-approximate the exact intermediate reachable set. For each new predicate variable, we add 3 new constraints.

One can see that determining where splits occur is crucial in the exact and over-approximate analysis of a ReLU layer. To do this, we need to know the ranges of all pixels in the ImageStar input set. However, as mentioned earlier, the computation of the exact range is expensive. To reduce the computation cost, we first use the estimated ranges of all pixels to remove a vast amount of non-splitting pixels. Then, we compute exact ranges for the pixels where splits may occur to compute the exact or over-approximate reachable set of the layer.

4.7 Reachabilty algorithm and parallelization

We have presented the core ideas for reachability analysis of different types of layers in a CNN. The reachable set of a CNN is constructed layer-by-layer in which the output reachable set of the previous layer is the input for the next layer. For the convolutional layer, average pooling layer and fully connected layer, we always can compute efficiently the exact reachable set of each layer. For the max pooling layer and ReLU layer, we can compute both the exact and the over-approximate reachable sets. However, the number of ImageStars in the

exact reachable set may grow quickly. Therefore, ***in the exact analysis, a layer may receive multiple input sets which can be handled in parallel to speed up the computation time***. The reachability algorithm for a CNN is summarized in Algorithm 1. The detailed implementation of the reachability algorithm for each layer can be found in NNV [34,39].

5 Evaluation

The proposed reachability algorithms are implemented in NNV [39], a tool for verification of deep neural networks and learning-enabled CPS. NNV utilizes core functions in MatConvNet [40] for the analysis of several layers. The evaluation of our approach consists of two parts. First, we evaluate robustness verification of deep neural networks in comparison to zonotope [29] and polytope methods [30] that are re-implemented in NNV. Second, we evaluate the scalability of our approach and the DeepPoly polytope method using real-world image classifiers, VGG16, and VGG19 [28]. The experiments are done on a computer with following configurations: Intel Core i7-6700 CPU @ 3.4GHz × 8 Processor, 62.8 GiB Memory, Ubuntu 18.04.1 LTS OS.[5] Lastly, we present a comparison with ERAN-DeepZ method on their $ConvMaxPool$ network trained on the CIFAR-10 data set in the Appendix Table 6.

5.1 Robustness Verification of MNIST Classification Networks

We compare our approach with the zonotope and polytope methods in two aspects including verification time and conservativeness of the results. To do that, we train 3 CNNs in small, medium, and large architectures with 98%, 99.7%, and 99.9% accuracy, respectively, using the MNIST data set consisting of 60000 images of handwritten digits with a resolution of 28×28 pixels [20]. The network architectures are given in the Appendix Figure 13.

The networks classify images into ten classes: $0, 1, \ldots, 9$. The classified output is the index of the dimension that has maximum value, i.e., the argmax across the 10 outputs. We evaluate the robustness of the network under the well-known brightening attack used in [8]. The idea of a brightening attack is that we can change the value of some pixels independently in the image to make it brighter or darker to fool the network, to misclassify the image. In this case study, we darken a pixel of an image if its value x_i (between 0 and 255) is larger than a threshold d, i.e., $x_i \geq d$. Mathematically, we reduce the value of that pixel x_i to the new value x_i' such that $0 \leq x_i' \leq \delta \times x_i$.

The robustness verification is done as follows. We select 100 images that are correctly classified by the networks and perform the brightening attack on these, which are then used to evaluate the robustness of the networks. A network is

[5] Comparison code is available in the NNV repository: https://github.com/verivital/nnv/tree/cav2020imagestar/code/nnv/examples/Submission/CAV2020_ImageStar and as a CodeOcean capsule [34]: https://doi.org/10.24433/CO.3351375.v1.

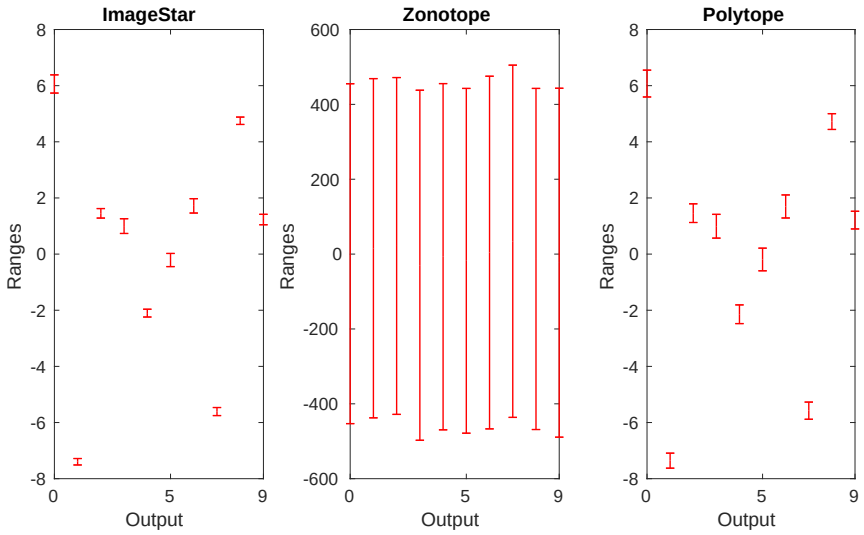

Fig. 8: Example output ranges of the small MNIST classification network using different approaches.

robust to an input set if, for any *attacked* image, this is correctly classified by the network. We note that the input set contains an infinite number of images. Therefore, to prove the robustness of the network to the input set, we first compute the output set containing all possible output vectors of the network using reachability analysis. Then, we prove that in the output set, the correctly classified output always has the maximum value compared with other outputs. Note that we can neglect the *softmax* and *classoutput* layers of the networks in the analysis since we only need to know the maximum output in the output set of the last fully connected layer in the networks to prove the robustness of the network.

We are interested in the percentage of the number of input sets that a network is provably robust and the verification times of different approaches under different values of d and θ. When d is small, the number of pixels in the image that are attacked is large and vice versa. For example, the average number of pixels attacked (computed on 100 cases) corresponding to $d = 250$, 245 and 240 are 15, 21 and 25 respectively. The value of δ dictates the size of the input set that can be created by a specific attack. Stated differently it dictates the range in which the value of a pixel can be changed. For example, if $d = 250$ and $\delta = 0.01$, the value of an attacked pixel many range from 0 to 2.55.

The experiments show that using the zonotope method, we cannot prove the robustness of any network. The reason is that the zonotope method obtains very conservative reachable sets. Figure 8 illustrates the ranges of the outputs computed by our ImageStar (approximate scheme), the zonotope and polytope approaches when we attack a digit 0 image with brightening attack in which $d = 250$ and $\delta = 0.05$. One can see that, using ImageStar and polytope method, we can prove that the output corresponding to the digit 0 is the one that has a maximum value, which means that the network is robust in this case. However,

	Robustness Results (in Percent)					
	$\delta = 0.005$		$\delta = 0.01$		$\delta = 0.015$	
	Polytope	*ImageStar*	*Polytope*	*ImageStar*	*Polytope*	*ImageStar*
$d = 250$	86.00	87.00	84.00	87.00	83.00	87.00
$d = 245$	77.00	78.00	72.00	78.00	70.00	77.00
$d = 240$	72.00	73.00	67.00	72.00	65.00	71.00
	Verification Times (in Seconds)					
$d = 250$	11.24	16.28	18.26	28.19	26.42	53.43
$d = 245$	14.84	19.44	24.96	40.76	38.94	85.97
$d = 240$	18.29	25.77	33.59	64.10	54.23	118.58

Table 1: Verification results of the small MNIST CNN.

	Robustness Results (in Percent)					
	$\delta = 0.005$		$\delta = 0.01$		$\delta = 0.015$	
	Polytope	*ImageStar*	*Polytope*	*ImageStar*	*Polytope*	*ImageStar*
$d = 250$	86.00	99.00	73.00	99.00	65.00	99.00
$d = 245$	74.00	95.00	58.00	95.00	46.00	95.00
$d = 240$	69.00	90.00	49.00	89.00	38.00	88.00
	Verification Times (in Seconds)					
$d = 250$	213.86	52.09	627.14	257.12	1215.86	749.41
$d = 245$	232.81	68.98	931.28	295.54	2061.98	1168.31
$d = 240$	301.58	102.61	1451.39	705.03	3148.16	2461.89

Table 2: Verification results of the medium MNIST CNN.

the zonotope method produces very large output ranges that cannot be used to prove the robustness of the network. The figure also shows that our ImageStar method produces tighter ranges than the polytope method, which means our result is less conservative than the one obtained by the polytope method. We note that the zonotope method is very time-consuming. It needs 93 seconds to compute the reachable set of the network in this case, while the polytope method only needs 0.3 seconds, and our approximate ImageStar method needs 0.74 seconds. The main reason is that the zonotope method introduces many new variables when constructing the reachable set of the network, which results in the increase in both computation time and conservativeness.

The comparison of the polytope and our ImageStar method is given in Tables 1, 2, and 3. The tables show that in all networks, our method is less conservative than the polytope approach since the number of cases that our approach can prove the robustness of the network is larger than the one proved by the polytope method. For example, for the small network, for $d = 240$ and $\delta = 0.015$, we can prove 71 cases while the polytope method can prove 65 cases. Importantly, the number of cases proved by DeepPoly reduces quickly when the network becomes larger. For example, for the case that $d = 240$ and $\delta = 0.015$, the polytope method is able to prove the robustness of the medium network for 38 cases while our approach can prove 88 cases. This is because the polytope method becomes more and more conservative when the network or the input set is large. The tables show that the polytope method is faster than our ImageStar method on the small network. However, it is slower than the ImageStar method on any larger

	Robustness Results (in Percent)					
	$\delta = 0.005$		$\delta = 0.01$		$\delta = 0.015$	
	Polytope	*ImageStar*	*Polytope*	*ImageStar*	*Polytope*	*ImageStar*
$d = 250$	90.00	99.00	83.00	99.00	*MemErr*	99.00
$d = 245$	91.00	100.00	75.00	100.00	*MemErr*	100.00
$d = 240$	81.00	99.00	*MemErr*	99.00	*MemErr*	99.00
	Verification Times (in Seconds)					
$d = 250$	917.23	67.45	5221.39	231.67	*MemErr*	488.69
$d = 245$	1420.58	104.71	6491.00	353.02	*MemErr*	1052.87
$d = 240$	1872.16	123.37	*MemErr*	476.67	*MemErr*	1522.50

Table 3: Verification results of the large MNIST CNN.

	Robustness Results (in percentage)							
	VGG16				**VGG19**			
	$\delta = 10^{-7}$		$\delta = 2 \times 10^{-7}$		$\delta = 10^{-7}$		$\delta = 2 \times 10^{-7}$	
	Polytope	*ImageStar*	*Polytope*	*ImageStar*	*Polytope*	*ImageStar*	*Polytope*	*ImageStar*
$l = 0.96$	85.00	85.00	85.00	85.00	100.00	100.00	100.00	100.00
$l = 0.97$	85.00	85.00	85.00	85.00	100.00	100.00	100.00	100.00
$l = 0.98$	85.00	85.00	85.00	85.00	95.00	95.00	95.00	95.00
	Verification Times (in Seconds)							
$l = 0.96$	319.04	318.60	327.61	319.93	320.91	314.14	885.07	339.30
$l = 0.97$	324.93	323.41	317.27	324.90	315.84	315.27	319.67	314.58
$l = 0.98$	315.54	315.26	468.59	332.92	320.53	320.44	325.92	317.95

Table 4: Verification results of VGG networks.

networks in all cases. Notably, for the large network, the ImageStar approach is significantly faster than the polytope approach, 16.65 times faster in average. The results also show that the polytope approach may run into memory problem for some large input sets.

5.2 Robustness Verification of VGG16 and VGG19

In this section, we evaluate the polytope and ImageStar methods on real-world CNNs, the VGG16 and VGG19 classification networks [28]. We use Foolbox [26] to generate the well-known DeepFool adversarial attacks [25] on a set of 20 bell pepper images. From an original image ori_im, Foolbox generates an adversarial image adv_im that can fool the network. The difference between two images is defined by $diff\_im = adv\_im - ori\_im$. We want to verify if we apply $(l + \delta)$ percent of the attack on the original image, whether or not the network classifies the disturbed images correctly. The set of disturbed images can be represented as an ImageStar as follows $disb\_im = ori\_im + (l + \delta) \times diff\_im$, where l is the percentage of the attack at which we want to verify the robustness of the network, and δ is a small perturbation around l, i.e., $0 \leq \delta \leq \delta_{max}$. Intuitively, l describes how close we are to the attack, and the perturbation δ represents the size of the input set.

Table 4 shows the verification results of VGG16 and VGG19 with different levels of the DeepFool attack. The networks are robust if they classify correctly the set of disturbed images $disb\_im$ as bell peppers. To guarantee the robustness

l	δ_{max}	VGG16				VGG19			
		Exact		Approximate		Exact		Approximate	
		Robust	VT	Robust	VT	Robust	VT	Robust	VT
50%	10^{-7}	Yes	64.56226	Yes	60.10607	Yes	234.11977	Yes	72.08723
	2×10^{-7}	Yes	63.88826	Yes	59.48936	Yes	1769.69313	Yes	196.93728
80%	10^{-7}	Yes	64.92889	Yes	60.31394	Yes	67.11730	Yes	63.33389
	2×10^{-7}	Yes	64.20910	Yes	59.77254	Yes	174.55983	Yes	200.89500
95%	10^{-7}	Yes	67.64783	Yes	59.89077	Yes	73.13642	Yes	67.56389
	2×10^{-7}	Yes	63.83538	Yes	59.23282	Yes	146.16172	Yes	121.91447
97%	10^{-7}	Yes	64.30362	Yes	59.79876	Yes	77.25398	Yes	64.43168
	2×10^{-7}	Yes	64.06285	Yes	61.23296	Yes	121.70296	Yes	107.17331
98%	10^{-7}	Yes	64.06183	Yes	59.89959	No	67.68139	Unkown	64.47035
	2×10^{-7}	Yes	64.01997	Yes	59.77469	No	205.00939	Unknown	107.42679
98.999%	10^{-7}	Yes	64.24773	Yes	60.22833	No	71.90568	Unknown	68.25916
	2×10^{-7}	Yes	63.67108	Yes	59.69298	No	106.84492	Unknown	101.04668

Table 5: Verification results of the VGG16 and VGG19 in which VT is the verification time (in seconds) using the ImageStar exact and approximate schemes.

of the networks, the output corresponding to the bell pepper label (index 946) needs to be the maximum output compared with others. The table shows that with a small input set, small δ, the polytope and ImageStar can prove robustness of VGG16 and VGG19 in a reasonable amount of time. Notably, the verification times as well as the robustness results of the polytope and ImageStar methods are similar when they deal with small input sets except for two cases where ImageStar is faster than the polytope method. It is interesting to note that according to the verification results for the VGG and MNIST networks, deep networks may be more robust than shallow networks.

5.3 Exact Analysis vs. Approximate Analysis

We compare our ImageStar approximate scheme with the zonotope and polytope approximation methods, and investigate the performance of the ImageStar exact scheme compared to the approximate one. To illustrate the advantages and disadvantages of the exact scheme and approximate scheme, we consider the robustness verification of VGG16 and VGG19 on a single ImageStar input set created by an adversarial attack on a bell pepper image. The verification results are presented in Table 5. The table shows that for a small perturbation δ, the exact and over-approximate analysis can prove the robustness of the VGG16 around some specific levels of attack in approximately one minute. We can intuitively verify the robustness of the VGG networks via visualization of their output ranges. An example of the output ranges of VGG19 for the case of $l = 0.95\%, \delta_{max} = 2 \times 10^{-7}$ is depicted in Figure 9. One can see from the figure that the output of the index 946 corresponding to the bell pepper label is always the maximum one compared with others, which proves that VGG19 is robust in this case. From the table, it is interesting that VGG19 is not robust if we apply $\geq 98\%$ of the attack. Notably, the exact analysis can give us correct answers with a counter-example set in this case. However, the over-approximate analysis cannot prove that VGG19 is not robust since its obtained reachable set

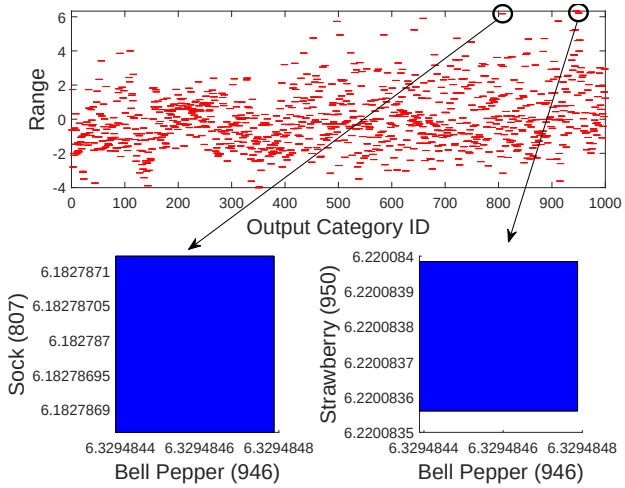

Fig. 9: Exact ranges of VGG19 show that VGG19 correctly classifies the input image as a bell pepper.

is an over-approximation of the exact one. Therefore, it may be the case that the over-approximate reachable set violates the robustness property because of its conservativeness. A counter-example generated by the exact analysis method is depicted in Figure 10 in which the disturbed image is classified as strawberry instead of bell pepper since the strawberry output is larger than the bell pepper output in this case.

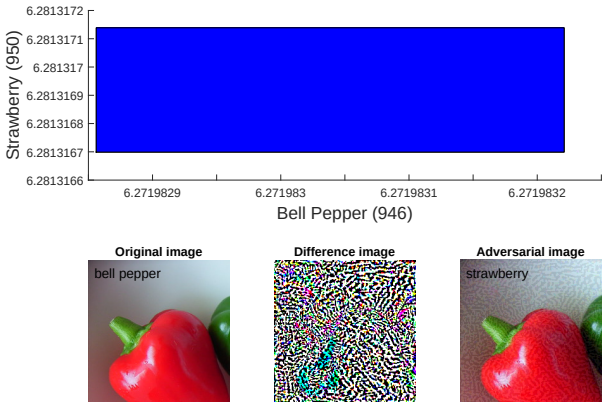

Fig. 10: A counter-example shows that VGG19 misclassifies the input image as a strawberry instead of a bell pepper.

To optimize the verification time, it is important to know the times consumed by each type of layer in the reachability analysis step. Figure 11 described the total reachability times of the convolutional layers, fully connected layers, max pooling layers and ReLU layers in the VGG19 with 50% attack and 10^{-7} per-

turbation. As shown in the figure, the reachable set computation in the convolutional layers and fully connected layers can be done very quickly, which shows the advantages of the ImageStar data structure. Notably, the total reachability time is dominated by the time of computing the reachable set for 5 max pooling layers and 18 ReLU layers. This is because the computation in these layers concerns solving a large number of linear programing (LP) optimization problems such as finding lower bound and upper bound, and checking max point candidates. Therefore, to optimize the computation time, we need to minimize the number of LP problems in the future.

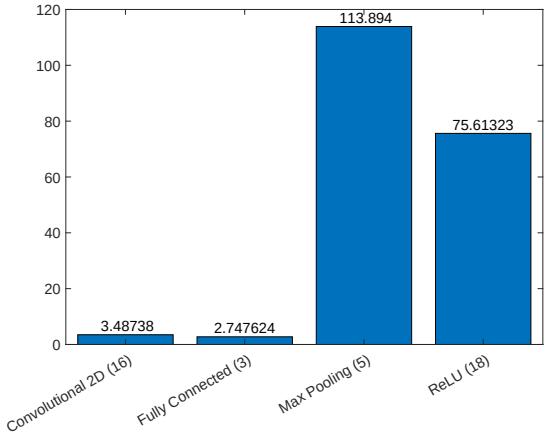

Fig. 11: Total reachability time of each type of layer in VGG19, where the max pooling and ReLU layers dominate the total reachability time.

6 Discussion

When we apply our approach on large networks, it has been shown that the size of the input set is the most important factor that influences the performance of verification approaches. However, this important issue has not been emphasized in the existing literature. Most of existing approaches focus on the size of the network that they can analyze. We hypothesize that existing methods (including the methods in this paper) scalable to large networks are only so for small input sets. When the input set is large, it causes three major problems in the analysis, which are explosions of 1) computation time; 2) memory usage; and 3) conservativeness. In the exact analysis method, a large input set causes more splits in the max-pooling and ReLU layers. A single ImageStar may split into many new ImageStars after these layers, which leads to explosion in the number of ImageStars in the reachable set as shown in Figure 12. Therefore, it requires more memory to handle the new ImageStars and more time for the computation. One may think that the over-approximate method can overcome this challenge since it obtains only one ImageStar at each layer and at the cost of conservativeness of the result. An over-approximate method does usually help reduce the computation time, as shown in the experimental results. However, it is not

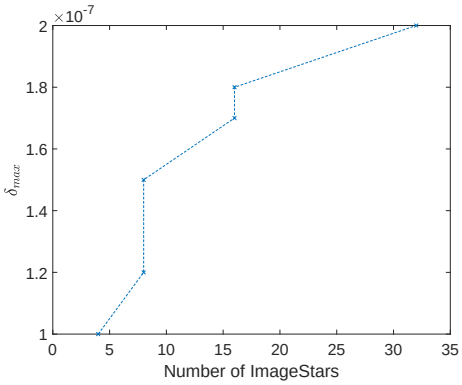

Fig. 12: Number of ImageStars in exact analysis increases with input size.

necessarily efficient in terms of memory consumption. The reason is, if there is a split, it introduces a new predicate variable and new generator. If the number of generators and the dimensions of the ImageStar are large, it requires a massive amount of memory to store the over-approximate reachable set. For instance, if there are 100 splits in the first ReLU layer of VGG19, the second convolutional layer will receive an ImageStar of size $224 \times 224 \times 64$ with 100 generators. To store this ImageStar with double precision, we need approximately $2.4GB$ of memory. In practice, the dimensions of the ImageStars obtained in the first several convolutional layers are usually large. Therefore, if splitting happens in these layers, we may run out of memory. We see that existing approaches, such as those using zonotopes and polytopes, also face the same challenges. Additionally, the conservativeness of an over-approximate reachable set is a crucial factor in evaluating an over-approximation approach. Therefore, the exact analysis still plays an essential role in the analysis of neural networks since it helps to evaluate the conservativeness of the over-approximation approaches.

7 Conclusion

We have presented a new set-based method for robustness verification of deep CNNs using ImageStars. The core of this method are exact and over-approximate reachability algorithms for ImageStar input sets. The experiments show that our approach is less conservative than recent zonotope and polytope approaches. It is also faster than existing approaches when dealing with deep networks. Notably, our approach can be applied to verify the robustness of real-world CNNs with small perturbed input sets. It can also compute the exact reachable set and visualize the exact output range of deep CNNs, and the analysis can speed up significantly with parallel computing. We have found and shown the size of the input set to be an important factor that impacts the performance of reachability algorithms. Future work includes improving the method to deal with larger input sets and optimizing the memory and time complexity of our computations.

References

1. Akintunde, M.E., Botoeva, E., Kouvaros, P., Lomuscio, A.: Formal verification of neural agents in non-deterministic environments. In: Autonomous Agents and Multi-Agent Systems (May 2020)
2. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 731–744. PLDI 2019, Association for Computing Machinery, New York, NY, USA (2019)
3. Bak, S., Duggirala, P.S.: Simulation-equivalent reachability of large linear systems with inputs. In: International Conference on Computer Aided Verification. pp. 401–420. Springer (2017)
4. Bak, S., Tran, H.D., Johnson, T.T.: Numerical verification of affine systems with up to a billion dimensions. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 23–32. ACM (2019)
5. Dutta, S., Jha, S., Sanakaranarayanan, S., Tiwari, A.: Output range analysis for deep neural networks. arXiv preprint arXiv:1709.09130 (2017)
6. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI. pp. 550–559 (2018)
7. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Narayan Kumar, K. (eds.) Automated Technology for Verification and Analysis. pp. 269–286. Springer International Publishing (2017)
8. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18. IEEE (2018)
9. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
10. Gopinath, D., Converse, H., Pasareanu, C., Taly, A.: Property inference for deep neural networks. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 797–809 (Nov 2019)
11. Hein, M., Andriushchenko, M.: Formal guarantees on the robustness of a classifier against adversarial manipulation. In: Advances in Neural Information Processing Systems. pp. 2266–2276 (2017)
12. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: Reachnn: Reachability analysis of neural-network controlled systems. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–22 (2019)
13. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Case study: verifying the safety of an autonomous racing car with a neural network controller. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control. pp. 1–7 (2020)
14. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Hybrid Systems: Computation and Control (HSCC) (2019)
15. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
16. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and

analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
17. Kouvaros, P., Lomuscio, A.: Formal verification of cnn-based perception systems. arXiv preprint arXiv:1811.11373 (2018)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
19. Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D.: Face recognition: A convolutional neural-network approach. IEEE transactions on neural networks **8**(1), 98–113 (1997)
20. LeCun, Y.: The MNIST database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
21. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
22. Lin, W., Yang, Z., Chen, X., Zhao, Q., Li, X., Liu, Z., He, J.: Robustness verification of classification deep neural networks via linear programming. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11418–11427 (2019)
23. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint arXiv:1706.07351 (2017)
24. Lopez, D.M., Musau, P., Tran, H.D., Johnson, T.T.: Verification of closed-loop systems with neural network controllers. In: Frehse, G., Althoff, M. (eds.) ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 61, pp. 201–210. EasyChair (April 2019)
25. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2574–2582 (2016)
26. Rauber, J., Brendel, W., Bethge, M.: Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. arXiv preprint arXiv:1707.04131 **5** (2017)
27. Ruan, W., Wu, M., Sun, Y., Huang, X., Kroening, D., Kwiatkowska, M.: Global robustness evaluation of deep neural networks with provable guarantees for the l_0 norm. arXiv preprint arXiv:1804.05805 (2018)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
29. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems. pp. 10825–10836 (2018)
30. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 41 (2019)
31. Souradeep Dutta, Xin Chen, S.S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Hybrid Systems: Computation and Control (HSCC) (2019)
32. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Hybrid Systems: Computation and Control (HSCC) (2019)
33. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. arXiv preprint arXiv:2004.05511 (2020)

34. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars (CodeOcean Capsule). https://doi.org/10.24433/CO.3351375.v1 (May 2020)
35. Tran, H.D., Cei, F., Lopez, D.M., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. In: ACM SIGBED International Conference on Embedded Software (EMSOFT'19). ACM (October 2019)
36. Tran, H.D., Musau, P., Lopez, D.M., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Parallelizable reachability analysis algorithms for feed-forward neural networks. In: 7th International Conference on Formal Methods in Software Engineering (FormaliSE2019), Montreal, Canada (2019)
37. Tran, H.D., Musau, P., Lopez, D.M., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analsysis for deep neural networks. In: 23rd International Symposisum on Formal Methods (FM'19). Springer International Publishing (October 2019)
38. Tran, H.D., Nguyen, L.V., Hamilton, N., Xiang, W., Johnson, T.T.: Reachability analysis for high-index linear differential algebraic equations (daes). In: 17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'19). Springer International Publishing (August 2019)
39. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: 32nd International Conference on Computer-Aided Verification (CAV) (July 2020)
40. Vedaldi, A., Lenc, K.: Matconvnet: Convolutional neural networks for matlab. In: Proceedings of the 23rd ACM international conference on Multimedia. pp. 689–692. ACM (2015)
41. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. arXiv preprint arXiv:1804.10829 (2018)
42. Weng, T.W., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Boning, D., Dhillon, I.S., Daniel, L.: Towards fast computation of certified robustness for relu networks. arXiv preprint arXiv:1804.09699 (2018)
43. Wong, E., Kolter, J.Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. arXiv preprint arXiv:1711.00851 (2017)
44. Wu, M., Wicker, M., Ruan, W., Huang, X., Kwiatkowska, M.: A game-based approximate verification of deep neural networks with provable guarantees. Theoretical Computer Science (2019)
45. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE Transactions on Neural Networks and Learning Systems **29**(11), 5777–5783 (2018)
46. Xiang, W., Tran, H.D., Yang, X., Johnson, T.T.: Reachable set estimation for neural network control systems: A simulation-guided approach. IEEE Transactions on Neural Networks and Learning Systems pp. 1–10 (2020)
47. Xiang, W., Tran, H.D., Johnson, T.T.: Reachable set computation and safety verification for neural networks with relu activations. arXiv preprint arXiv:1712.08163 (2017)
48. Xiang, W., Tran, H.D., Johnson, T.T.: Specification-guided safety verification for feedforward neural networks. AAAI Spring Symposium on Verification of Neural Networks (2019)
49. Xiang, W., Tran, H.D., Rosenfeld, J.A., Johnson, T.T.: Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. arXiv preprint arXiv:1802.06981 (2018)

50. Yang, X., Tran, H.D., Xiang, W., Johnson, T.T.: Reachability analysis for feed-forward neural networks using face lattices. https://arxiv.org/abs/2003.01226 (2020)
51. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Advances in Neural Information Processing Systems. pp. 4944–4953 (2018)

A Appendix

A.1 Exact reachability algorithm for a max-pooling layer

Algorithm 2 illustrates the exact reachability of a max-pooling layer with noticing that for an ImageStar set I, the anchor and generator images are put into a single 4-dimensional array V called *basis array* in which $V(:,:,:,1)$ is the anchor images. The algorithm works as follows. Firstly, we perform zero-padding for the ImageStar input set I (line 2) by padding zeros to the anchor and generator images. Secondly, the zero-padding set I' is used to compute the size of the *max map* $[h, w]$ and the start indexes $startID$ of each local region. Thirdly, we initialize the basis V' of the max map (line 6) and find all max-point candidates $maxID$ for every single point $[i, j, k]$ in the max map (line 13). If there is only one max-point candidate for a point $[i, j, k]$ in the max map, we update the basis of the max map. If not, we store this point to a list of splitting points $splitID$ (line 18) and then initialize the max map R (line 19). Lastly, we perform splitting operations through the list of splitting points (line 23) to get the final output set which is an array of ImageStars.

A.2 Approximate reachability algorithm for a max-pooling layer

Algorithm 3 illustrates the approximate reachability of a max-pooling layer. Similar to the exact algorithm, we perform zero-padding for the ImageStar input set I (line 2) by padding zeros to the anchor and generator images. The zero-padding set I' is then used to compute the size of the *max map* $[h, w]$ and the start indexes $startID$ of each local region. Thirdly, we initialize the basis V' of the max map (line 6) and find all max-point candidates $maxID$ for every single point $[i, j, k]$ in the max map (line 13). We use np to count the number of predicate variables when we do over-approximate reachability (line 15). If a local region has more than one max-point candidate, we introduce a new predicate variable representing the max-point of that region. Using the max-point candidate indexes $maxID$ and the new predicate variable index new_pred_id, we update the basis of the max map (line 17-28). Finally, we add the constraints on the new introduced predicate variables (line 30-57) and then construct the over-approximate ImageStar output set R (line 58).

A.3 Architectures of MNIST networks (Figure 13)

A.4 Comparison with ERAN-DeepZ [29]

In this section, we present the comparison between NNV and ERAN-DeepZ on the ERAN $ConvMaxPool$ network on a subset of CIFAR-10 data set containing 60000 32×32 color images in 10 classes.[6] We choose this network because it is the only one that has max-pooling layers in all ERAN networks trained on CIFAR-10. The network has 4 Convolutional layers, 2 max-pooling layer, 7 relu

[6] https://github.com/verivital/run_nnv_eran_comparison

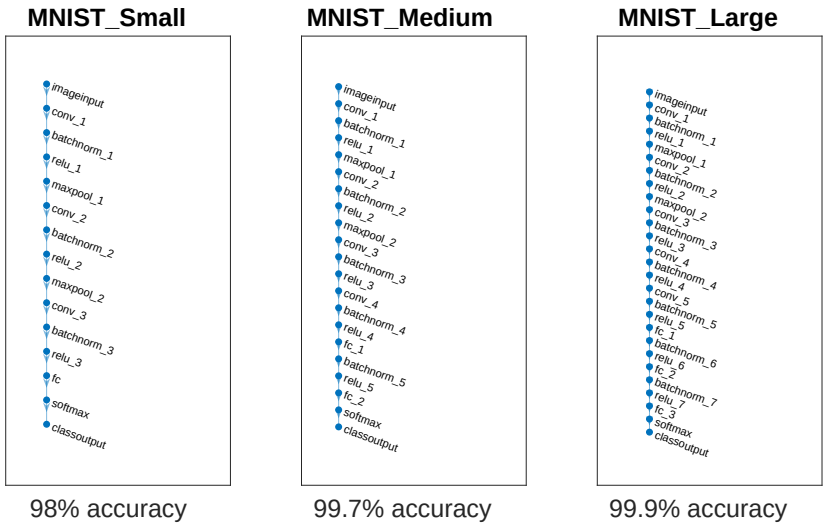

Fig. 13: The architectures of the small, medium, and large MNIST classification networks.

layers, and 3 fully connected layers. Although ERAN has chosen 100 images for the robustness analysis, we have experienced that only 48 images are correctly classified by the network without any attacks which means the accuracy of the network is very low ($\approx 48\%$). Despite the low accuracy, we still use it to clarify the advantages/disadvantages of our ImageStar approach in comparison with ERAN-DeepZ. We note that the pixel values in the tested images are scaled into [0, 1] in the analysis, and the well-known brightening attack [8] is used for the robustness analysis of the network. For any image, the values of some pixels x_i are changed independently to x_i' under the brightening attack which results in a set of images that can be described by a zonotope $Z = \{x_i' | 1 - \delta \leq x_i \leq x_i' \leq 1 \ \vee x_i' = x_i\}$ where δ is called a robustness bound [8].

The robustness verification results are presented in Table 6, which shows that our method gives a slightly better result than ERAN in terms of number of verifiable images. Additionally, the verification time of our approach grows very quickly as the robustness value δ increases. The growth of verification time is the cost we pay to improve the conservativeness of the over-approximate methods. One can see that the ERAN-DeepZ method is faster than the ImageStar method when dealing with large input sets (e.g., more attacked pixels) since it does not solve any LP optimization problems when constructing the reachable sets. In the future, we plan to develop a more relaxed ImageStar method that can neglect solving LP optimization problems to handle with larger input sets.

Robustness Results (in Percent)									
$\delta = 0.005$		$\delta = 0.008$		$\delta = 0.01$		$\delta = 0.012$		$\delta = 0.015$	
ERAN	*ImageStar*	*ERAN*	*ImageStar*	*ERAN*	*ImageStar*	*ERAN*	*ImageStar*	*ERAN*	*ImageStar*
48/48	48/48	46/48	47/48	46/48	47/48	44/48	46/48	44/48	46/48
Verification Times (in Seconds)									
115	78.05	122	155.45	123	153.67	130	454.44	129	468.49

Table 6: Verification results of the CIFAR-10 $ConvMaxPool$ network.

Algorithm 2 Exact reachability algorithm for a max pooling layer.

procedure $R = reach\_exact(I = \langle V, C, P\rangle, poolSize, stride, paddingSize)$
  $I' = zero\_padding(I, paddingSize)$ ▷ zero padding for the input set
  $[h, w] = get\_size\_maxMap(I', poolSize, stride)$ ▷ compute the size of the max map
  $nc = I'.number\_channels$, ▷ get the number of channels
  $np = I'.number\_predicate\_variables$ ▷ get the number of predicate variables
  $V'(:, :, nc, np + 1) = zeros(h, w)$ ▷ pre-allocate the basis of the max map
  $startID = get\_startPoints(I', poolSize, stride)$ ▷ the start index for each local region
  $maxID = cell(h, w, nc)$, ▷ to store the index of the max-point candidates
  $splitID = [\,]$
  for $k = 1 : nc$ **do**
    for $i = 1 : h$ **do**
      for $j = 1 : w$ **do**
        $maxID\{i, j, k\} = I'.get\_localMax\_index(startID\{i, j\}, poolSize, k)$
        if $size(maxID\{i, j, k\}, 1) == 1$ **then**
          $[i'\ j'\ k] = maxID\{i, j, k\}$ ▷ the local region has only one max-point
          $V'(i, j, k, :) = I'.V(i', j', k, :)$ ▷ update the basis of the max map
        else
          $[i\ j\ k] \rightarrow splitID$ ▷ store splitting index
  $R = \langle V', C, P\rangle$
  $n = size(splitID, 1)$ ▷ number of splitting indexes
  for $l = 1 : n$ **do**
    $[i\ j\ k] = splitID(l, :, :)$ ▷ get splitting index
    $R = split(R, I', [i\ j\ k], maxID\{i, j, k\})$ ▷ split ImageStars
procedure $R' = split(R, I', [i\ j\ k], maxID\{i, j, k\})$
  $R = [R_1\ R_2\ \ldots\ R_m]$ ▷ multiple input sets
  $R' = [\,]$
  for $l = 1 : m$ **do**
    $IS = stepSplit(R_l, I', [i\ j\ k], maxID\{i, j, k\})$ ▷ step splitting
    $IS \rightarrow R'$ ▷ store the new ImageStars
procedure $R' = stepSplit(R_l, I', [i\ j\ k], maxID\{i, j, k\})$
  $maxID\{i, j, k\} = [i'_1\ j'_1\ k;\ \ldots\ ; i'_q\ j'_q\ k]$ ▷ the local region has q max-points
  $R' = [\,]$
  for $l = 1 : q$ **do** ▷ a single ImageStar is split into q ImageStars
    $max\_point = [i'_l\ j'_l\ k]$, $others = maxID\{i, j, k\} \setminus max\_point$
    $[C', d'] = getConstraints(R_l, I', max\_point, others)$ ▷ get the constraints on the predicate variables that make a max-point candidate become the max point
    $V' = R_l.V$, $V'(i, j, k, :) = I'(i'_l, j'_l, k, :)$ ▷ update the basis
    $IS = \langle V', C', d'\rangle$, $IS \rightarrow R'$ ▷ construct and store the reach set

Algorithm 3 Approximate reachability algorithm for a max pooling layer.

procedure $R = reach\_exact(I = \langle V, C, P\rangle, poolSize, stride, paddingSize)$
  $I' = zero\_padding(I, paddingSize)$ ▷ zero padding for the input set
  $[h, w] = get\_size\_maxMap(I', poolSize, stride)$ ▷ compute the size of the max map
  $nc = I'.number\_channels,$ ▷ get the number of channels
  $np_0 = I'.number\_predicate\_variables$ ▷ get the number of predicate variables
  $V'(:, :, nc, np + 1) = zeros(h, w)$ ▷ pre-allocate the basis of the max map
  $startID = get\_startPoints(I', poolSize, stride)$ ▷ the start index for each local region
  $maxID = cell(h, w, nc),$ ▷ to store the index of the max-point candidates
  $np = np_0$
  for $k = 1 : nc$ **do**
    for $i = 1 : h$ **do**
      for $j = 1 : w$ **do**
        $maxID\{i, j, k\} = I'.get\_localMax\_index(startID\{i, j\}, poolSize, k)$
        if $size(maxID\{i, j, k\}, 1) > 1$ **then**
          $np = np + 1$ ▷ increase the number of predicate variables
  ▽ update the basis of the max map
  $new\_pred\_id = 0$ ▷ new predicate variable index
  for $k = 1 : nc$ **do**
    for $i = 1 : h$ **do**
      for $j = 1 : w$ **do**
        if $size(maxID\{i, j, k\}, 1) == 1$ **then**
          for $p = 1 : np_0 + 1$ **do**
            $[i'\ j'\ k] = maxID\{i, j, k\}$
            $V'(i, j, k, p) = I'.V(i', j', k, p)$
        else
          $V'(i, j, k, 1) = 0$
          $new\_pred\_id = new\_pred\_id + 1$
          $V'(i, j, k, np_0 + 1 + new\_pre\_id) = 1$
  ▽ update the constraints on predicate variables
  $N = poolSize(1) \times poolSize(2)$
  $C' = zeros(new\_pre\_id \times (N + 1), np)$
  $d' = zeros(new\_pre\_id \times (N + 1), 1)$
  $l = 0$
  $\alpha = [\alpha_1 \dots \alpha_{np_0+l} \dots \alpha_{np}]^T$ ▽ vector of predicate variables
  for $k = 1 : nc$ **do**
    for $i = 1 : h$ **do**
      for $j = 1 : w$ **do**
        if $size(maxID\{i, j, k\}, 1) > 1$ **then**
          $l = l + 1$
          ▽ get all related local pixel indexes
          $points = I'.get\_localPoints(startIDi, j, poolSize, k)$
          ▽ upper bound of new predicate variable: $\alpha_{np_0+l} \le ub \equiv C_1\alpha \le d_1$
          $C_1 = zeros(1, np),\ C_1(np_0 + l) = 1$
          ▽ get bounds of the local pixel values
          $[lb, ub] = I'.get\_localBound(startIDi, j, poolSize, k)$
          $d_1 = ub$
          $C_2 = zeros(N, np),\ d_2 = zeros(N, 1)$
          for $g = 1 : N$ **do**
            $[i'\ j'] = points(g, :)$
            ▽ new constraint: $\alpha_l \ge x_{i'j'k} \equiv C_2\alpha \le d_2$
            $C_2(g, 1 : np_0) = I'.V(i', j', k, 2 : np_0 + 1),$
            $C_2(g, np_0 + l) = -1$
            $d_2(g) = -I'.V(i', j', k, 1)$
          $C'((l - 1) \times (N + 1) + 1 : l \times (N + 1), :) = [C_1; C_2]$
          $d'((l - 1) \times (N + 1) + 1 : l \times (N + 1)) = [d_1; d_2]$
  $C' = [I'.C\ \ zeros(size(I'.C, 1), l);\ C']$
  $d' = [I'.d;\ d']$
  $R = \langle V', C', d'\rangle$