CAPM: FAST AND ROBUST VERIFICATION ON MAXPOOL-BASED CNN VIA DUAL NETWORK

Jia-Hau Bai
Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan
r07942092@ntu.edu.tw

Chi-Ting Liu
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
b07901090@ntu.edu.tw

Yu Wang
Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan
r11942152@g.ntu.edu.tw

Fu-Chieh Chang
MediaTek Research, Taipei, Taiwan
Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan
d09942015@ntu.edu.tw

Pei-Yuan Wu
Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan
peiyuanwu@ntu.edu.tw

ABSTRACT

This study uses CAPM (Convex Adversarial Polytope for Maxpool-based CNN) to improve the verified bound for general purpose maxpool-based convolutional neural networks (CNNs) under bounded norm adversarial perturbations. The maxpool function is decomposed as a series of ReLU functions to extend the convex relaxation technique to maxpool functions, by which the verified bound can be efficiently computed through a dual network. The experimental results demonstrate that this technique allows the state-of-the-art verification precision for maxpool-based CNNs and involves a much lower computational cost than current verification methods, such as DeepZ, DeepPoly and PRIMA. This method is also applicable to large-scale CNNs, which previous studies show to be often computationally prohibitively expensive. Under certain circumstances, CAPM is 40-times, 20-times or twice as fast and give a significantly higher verification bound (CAPM 98% vs. PRIMA 76%/DeepPoly 73%/DeepZ 8%) as compared to PRIMA/DeepPoly/DeepZ. (cf. Fig. 3 and Fig. 4). Furthermore, we additionally present the time complexity of our algorithm as $O(W^2NK)$, where W is the maximum width of the neural network, N is the number of neurons, and K is the size of the maxpool layer's kernel.

1 INTRODUCTION

In the past few years, convolution neural networks have reached unprecedented performance in various tasks such as face recognition (Hu et al., 2015; Mehdipour Ghazi & Kemal Ekenel, 2016) and self-driving cars (Rao & Frtunikj, 2018; Maqueda et al., 2018), to name a few. However, these networks are vulnerable to malicious modification of the pixels in input images, known as adversarial examples, such as FGSM (Goodfellow et al., 2015), PGD (Madry et al., 2018), One Pixel Attack (Su

et al., 2019), Deepfool (Moosavi-Dezfooli et al., 2016), EAD (Chen et al., 2018), GAP (Poursaeed et al., 2018), MaF(Chaturvedi & Garain, 2020) and many others (Wong et al., 2019).

In view of the threat posed by adversarial examples, how to protect neural networks from being tricked by adversarial examples has become an emerging research topic. Previous studies of defense against adversarial examples are categorized as *Removal of adversarial perturbation* (Akhtar et al., 2018; Xie et al., 2019; Jia et al., 2019; Samangouei et al., 2018) and *Adversarial training* (Shafahi et al., 2019; Han et al., 2020; Tramer et al., 2020). Both defense mechanisms may protect the network from certain adversarial examples but there is only empirical evidence that they do so. The robustness of the network is not guaranteed. It is impossible to train or evaluate all possible adversarial examples so these methods are vulnerable to other adversarial examples that are not in the data sets that are used.

The need for guaranteed robustness assessments has led to the development of verification mechanisms for a neural network. These verify specific properties pertaining to neural networks, such as robustness against norm-bounded perturbation (Dvijotham et al., 2018; Singh et al., 2018), robustness against adversarial frequency or severity (Katz et al., 2017) and robustness against rotations (Singh et al., 2019a).

During the early development of neural network verification, satisfiability modulo theories (SMT) solver (Katz et al., 2017) and semidefinite programming (SDP) methods (Raghunathan et al., 2018) were used. A SMT solver yields tight verification bounds but is not scalable to contemporary networks with sophisticated architecture. The SDP method requires less time but is limited to linear architectures. Recent studies have developed verification tools for more realistic scenarios, such as a fully connected neural network (FCNN) with an activation function and a convolution neural network (CNN). As indicated by Salman et al. (2020), the main methods for neural network verification can be categorized as either primal view or dual view.

The primal view method involves *Abstract interpretation* and *Interval bound propagation*. There are classic frameworks for abstract interpretation (e.g., AI2 (Gehr et al., 2018), DeepZ (Singh et al., 2018), DeepPoly (Singh et al., 2019a)). As a step further, RefineZono (Singh et al., 2019b) and RefinePoly (Singh et al., 2019b) use mixed integer linear programming (MILP) to improve the verification bounds for DeepZ and DeepPoly, respectively. However, the computation time that is required for verification is significantly increased. Another bounding technique for the primal view method is interval bound propagation, which uses interval arithmetic to obtain the bound for each individual neuron in each layer. Representative works include IBP (Gowal et al., 2018) and CROWN-IBP (Zhang et al., 2019). Dual-view methods (Wong & Kolter, 2018; Dvijotham et al., 2018; Wong et al., 2018; Bunel et al., 2020; Xu et al., 2020; Wang et al., 2021) formulate the verification problem as an optimization problem, so according to Lagrangian duality (Boyd et al., 2004), each dual feasible solution yields a lower bound to the primal problem and verification bounds are derived by solving the dual problem. Moreover, noteworthy among these methods is the state-of-the-art approach α,β-CROWN (Wang et al., 2021), which is grounded in the LiRPA framework (Xu et al., 2020).

1.1 Verification of a CNN with maxpooling

The maxpool function is an integral part in most real-world neural network architectures, especially CNNs (e.g., LeNet (LeCun et al., 1998), AlexNet (Krizhevsky et al., 2012) and VGG (Simonyan & Zisserman, 2015)), which are widely used for image classification. However, past works have the following shortage in the verification of networks involving maxpool functions:

- *Not applicable:* IBP (Gowal et al., 2018) and others (Wong & Kolter, 2018; Wong et al., 2018) (Bunel et al., 2020; De Palma et al., 2021) verify a CNN but there is no theory to verify maxpool-based networks. Gowal et al. (2018) analyzed several monotonic activation functions (e.g., ReLU, tanh, sigmoid) in IBP but they did not consider non-monotonic functions (e.g., maxpool). Wong & Kolter (2018) discussed the verification of a ReLU-based FCNN and a later study (Wong et al., 2018) uses this for the verification of residual networks. However, besides referring to the work of Dvijotham et al. (2018), the study by Wong et al. (2018) does not address much about handling maxpool functions. Bunel et al. (2020); De Palma et al. (2021) analyzed networks with nonlinear activation functions, such as ReLU and sigmoid, but there is no analysis of the maxpool function.

- *Has theory but lack of implementation evidence:* These studies analyze the maxpool function but experiments only verifiy ReLU-based CNNs. Examples include AI2 (Gehr et al., 2018), DeepZ (Singh et al., 2018), DeepPoly (Singh et al., 2019a), RefineZono (Singh et al., 2019b), RefinePoly (Singh et al., 2019b) and LiRPA (Xu et al., 2020). Dvijotham et al. (2018) analyzed a large variety of activation functions, such as ReLU, tanh, sigmoid and maxpool, but they only demonstrated the experiment result on a small network consisting of one linear layer, followed by sigmoid and tanh. CROWN-IBP (Zhang et al., 2019) used the verification method of IBP (Gowal et al., 2018) and analyzed non-monotonic functions, including maxpool, but experiments only verify results for ReLU-based CNNs. In spite of providing functions corresponding to maxpool, LIRPA is currently unable to function properly on maxpool-based CNNs. The definitions of DenseNet and ResNeXt used in their experiments can be found in their GitHub repository, and it's noteworthy that these definitions do not include a maxpool layer.
- *Imprecise:* These studies give an imprecise verification bound for maxpool-based CNN. DeepZ (Singh et al., 2018), DeepPoly (Singh et al., 2019a), and PRIMA (Müller et al., 2022) implement the verification of maxpool-based CNN but experimental results are nevertheless lacking. For comparison purposes, we implement these methods on 6 maxpool-based CNN benchmarks modified from Mirman et al. (2018) (cf. Supplementary Material A.5). Our experiment indicates that DeepZ, DeepPoly, and PRIMA are imprecise in our benchmarks. For a norm-bounded perturbation $\epsilon = 0.0024$, the verified robustness for a convSmall CIFAR10 structure decreases to 1%, 25%, and 26%, respectively (cf. Fig. 3). Due to the implementation of the BaB (branch and bound) algorithm in α,β-CROWN (Wang et al., 2021) on maxpool-based CNNs, it results in excessive GPU memory requirements (more than 13GB) or prolonged execution times (more than 5 minutes per example). Consequently, we consider examples that trigger the BaB algorithm as not verified.
- *Computational costly:* These studies involve a significant computational cost to verify each input image in the maxpool-based CNN benchmarks (cf. Sec. 3). Our experiment shows that for $\epsilon = 0.0006$ on convBig CIFAR10 (cf. Fig. 4), PRIMA (Müller et al., 2022) requires 6.5 days and DeepPoly (Singh et al., 2019a) requires 3 days to verify 100 images. (See Sec. A.1.1 for hardware spec)

Yuan et al. (2019) showed that l_∞ norm is one of the most commonly used perturbation measurements (e.g.,DeepFool (Moosavi-Dezfooli et al., 2016), CW (Carlini & Wagner, 2017), Universal adversarial perturbations (Moosavi-Dezfooli et al., 2017), and MI-FGSM (Dong et al., 2018)). Therefore, this study verifies a maxpool-based CNN with l_∞ norm-bounded perturbation. The contributions of this study are:

- CAPM (Convex Adversarial Polytope for Maxpool-based CNN) is used to improve the verified bound for a maxpool-based CNN, assuming l_∞ norm-bounded input perturbations. The maxpool function is decomposed into multiple ReLU functions (see Sec. 2.2), as such convex relaxation trick by Wong & Kolter (2018) can be applied. While we are not the first to consider the relationship between maxpool functions and ReLU functions in our paper, our experiments demonstrate that the algorithm we propose is currently the most effective among all executable methods.
- A dual network for general purpose CNNs is derived with maxpool, padding and striding operations to obtain the verified bound efficiently (cf. (48)).
- The results (cf. Sec. 3) show that CAPM gives a verification that is as robust as the state-of-the-art methods (PRIMA), but there is significantly less runtime cost for MNIST and CIFAR 10.
- The experimental results show the limitations of a Monte-Carlo simulation for a large-scale neural network verification problem, which demonstrates the necessity for a provable robustness verification scheme.
- Among algorithms applicable to maxpool-based CNNs, we are currently the only ones providing a clearly defined time complexity for our algorithm. The time complexity of the CAPM algorithm is $O(W^2NK)$.

The main limitation of our algorithm is that it requires adherence to the assumptions defined in Supplementary Material. A.2.1. We address future work based on this in the conclusions section.

The reminder of this paper is organized as follows: Sec. 2 defines the verification problem and introduce the key idea of CAPM with a toy example. Sec. 3 compares the experimental result for CAPM against with DeepZ, DeepPoly, PRIMA, LiRPA and α,β-CROWN in terms of the verified robustness and average runtime metrics for various adversary budgets. The experiment result also indicates that the accuracy lower bound predicted by each verification method becomes looser as norm-bounded perturbation increases. Conclusions and future work are summarized in Sec. 4. Furthermore, we gives a bound analysis for intermediate layers to explain the reason for this phenomenon in Supplementary Material A.4.

2 METHOD

This section illustrates the verification of a CNN classifier under the l_∞ norm-bounded perturbations. Sec. 2.1 defines the verification problem and Sec. 2.2 gives an overview of CAPM. More details for solving the verification problem for a maxpool-based CNN are in Supplementary Material A.2 and Supplementary Material A.3.

Supplementary Material A.2 specifies the CNN architecture for this study; Supplementary Material A.3 describes the formulation of the the verification problem in terms of Lagranian dual problems, and simplifies the dual constraints to the form of a leaky ReLU dual network.

2.1 DEFINITION OF THE VERIFICATION PROBLEM

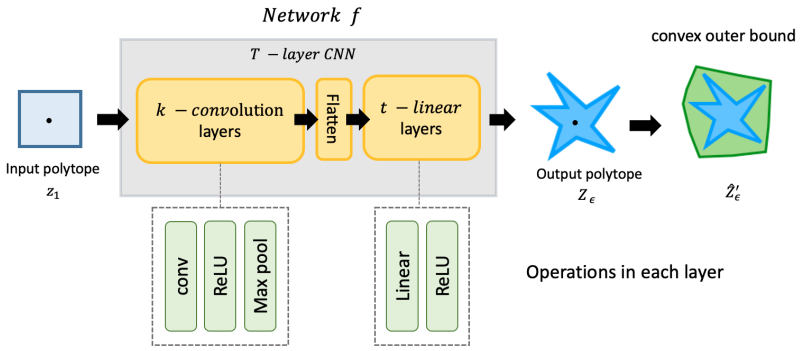

Figure 1: Adversarial polytope passes through network f

This study evaluates the robustness of a CNN to arbitrary adversarial examples within a bounded norm budget, to see whether or not the prediction result of the CNN will change under such adversarial perturbations. The verification problem is reformulated through convex relaxation as a convex optimization problem and the duality theorem states that any dual feasible solution can serve as a lower bound for the original verification problem.

A neural network f consisting of k convolution layers with ReLU activation and maxpool functions, followed by flattening and t fully connected layers with ReLU activation is shown in Fig 1. If a clean image x is added with perturbation Δ, to which this study impose a l_∞-norm constraint $\|\Delta\|_\infty \leq \epsilon$, the perturbed input z_1 resides in an input adversarial polytope that is described as:

$$\mathbf{z_1} \leq \mathbf{x} + \epsilon \quad \text{and} \quad \mathbf{z_1} \geq \mathbf{x} - \epsilon. \tag{1}$$

The perturbed input z_1 is taken as the input of the network f. Though at first glimpse the input adversarial polytope (cf. (1)) is a hyper-cube which is convex, and that the ReLU function is itself a convex function, the intermediate adversarial polytope after passing the input adversarial polytope through the convolution and ReLU activation is in general not a convex set, as the set $\{(\xi, ReLU(\xi)) : \xi \in \mathbb{R}\}$ is not a convex subset of $\mathbb{R}^2$. The verification problem deviates from a convex optimization framework due to the non-linearity of ReLU and maxpool equalities. The

presence of these non-linear equalities introduces complexity, rendering the optimization problem non-convex. Similarly, the application of a nonlinear function leads to a non-convex output polytope, even if the input polytope maintains convexity. In other words, substituting nonlinear equality constraints with linear inequality constraints results in the creation of a convex outer bound. To make the feasible solution a convex set, a convex outer bound (Wong & Kolter, 2018) is constructed when passing through each of the ReLU and maxpool functions. The output polytope Z_ϵ, which is the collection of all possible results computed by f at the output layer with a perturbed input z_1, is contained within a convex polytope $\hat{Z}'_\epsilon$. Both Z_ϵ and $\hat{Z}'_\epsilon$ are subsets of $\mathbb{R}^K$ for a K-class classification task.

For an image x that is labeled with ground truth $y^* \in \{1, ..., K\}$ to which an adversary attempts to mislead network f into falsely predicting a target label $y^{targ} \in \{1, ..., K\}$ rather than y^*, a necessary condition is that the adversary must find a perturbed input z_1 satisfying (1) so that $\mathbf{e}_{y^*}^T f(z_1) \leq \mathbf{e}_{y^{targ}}^T f(z_1)$, where $\mathbf{e}_{y^*}$ and $\mathbf{e}_{y^{targ}}$ are one-hot encoded vectors of y^* and y^{targ}, respectively. Therefore, as $f(z_1) \in Z_\epsilon \subset \hat{Z}'_\epsilon$, if the minimum for the optimization problem

$$\min_{\hat{\mathbf{y}} \in \hat{\mathbf{Z}}'_\epsilon} \quad (\mathbf{e_{y*}} - \mathbf{e_{y^{targ}}})^{\mathbf{T}} \hat{\mathbf{y}} \tag{2}$$

is positive for every target class $y^{targ} \in \{1, ..., K\} \setminus \{y^*\}$, then network f cannot be fooled by an adversarial example that differs from image x by a perturbation with at most ϵ under l_∞-norm. This method can guarantee zero false negatives, so the system flags every image that is prone to attack by an adversarial example, but it may falsely flag some images resilient to perturbations.

2.2 CAPM OVERVIEW

This section describes a toy example to illustrate the use of CAPM to solve the optimization problem in (2) using a Maxpool-based network. The dual problem is formulated using Lagrangian relaxation (Boyd et al., 2004) and convex relaxation (Wong & Kolter, 2018), so any dual feasible solution corresponds to a lower bound to the original problem in (2). Convex relaxation loosens the verification bound but Wong & Kolter (2018) showed that this lower bound can be calculated using a backpropagation-like dynamic programming process in a so called dual network. As the determination of upper and lower bounds for preceding layers constitutes a sub-problem within the dual network framework for subsequent layers, employing a dynamic programming algorithm becomes a viable approach to address this verification problem. This study extends the method of Wong & Kolter (2018) to a maxpool-based CNN and demonstrates that maxpool-based CNNs can also be verified efficiently and precisely using a dual network.

2.2.1 TOY EXAMPLE

CAPM verifies the robustness of a simple maxpool-based network under the l_∞ norm constraint in (1). The verification problem for this toy example is formulated as an optimization problem in (3). If the lower bound of (3) is positive for all possible target classes, then this network is not misled by any input perturbation, l_∞-norm that is less than ϵ. If not, then this network may not be safe for this input perturbation.

$$\begin{aligned}
\min_{\hat{\mathbf{z}}'_3} \quad & (\mathbf{e_{y*}} - \mathbf{e_{y^{targ}}})^{\mathbf{T}} \hat{\mathbf{z}}_{\mathbf{3}} \equiv \mathbf{d}^{\mathbf{T}} \hat{\mathbf{z}}_{\mathbf{3}} \\
\text{s.t.} \quad & \mathbf{z}_1 \leq \mathbf{x} + \epsilon \\
& \mathbf{z}_1 \geq \mathbf{x} - \epsilon \\
& \hat{\mathbf{z}}_2 = \mathbf{W}_1 \mathbf{z}_1 + \mathbf{b}_1 \\
& \mathbf{z}_2^R = \max(\hat{\mathbf{z}}_2,\ \mathbf{0}) \\
& \mathbf{z}_2 = \max(z_{2,0}^R,\ z_{2,1}^R,\ z_{2,2}^R,\ z_{2,3}^R) \\
& \hat{\mathbf{z}}_3 = \mathbf{W}_2 \mathbf{z}_2 + \mathbf{b}_2
\end{aligned} \tag{3}$$

In (3), the perturbed input $\mathbf{z}_1$ is the input to a simple maxpool-based network. The feature map $\hat{\mathbf{z}}_2$ is obtained by inputting $\mathbf{z}_1$ into the linear operation in the fully-connected layer. ReLU and maxpool are then used to compute the intermediate results $\mathbf{z}_2^R$ and $\mathbf{z}_2$. The output $\hat{\mathbf{z}}_3$ is calculated using the linear operation. This is a non-convex optimization problem because of the non-affine activation functions ReLU and maxpool so Wong & Kolter (2018)'s method of convex relaxation

(see Supplementary Material A.2.2 for more details) is applied to the ReLU function over the input interval, which approximates the ReLU function using the linear outer bounds (cf. Fig. 9). In terms of the maxpool function (see A.2.1 and A.2.2 for more details), $z_2 = \max(z^R_{2,0},\ z^R_{2,1},\ z^R_{2,2},\ z^R_{2,3})$ is decomposed into several one-by-one comparisons using dummy variables

$$z^M_{2,j+1} = \max(z^R_{2,j},\ z^M_{2,j}) = z^M_{2,j} + \max(z^R_{2,j} - z^M_{2,j},\ 0),\ j \in [\![0, 3]\!]. \tag{4}$$

As such, $z^M_{2,4} = \max(z^M_{2,0}, z^R_{2,0}, z^R_{2,1}, z^R_{2,2}, z^R_{2,3})$, and $z_2 = z^M_{2,4}$ if one chooses $z^M_{2,0}$ no larger than the maximum of $z^R_{2,0}, ..., z^R_{2,3}$. In this example, $z^M_{2,0} = 0$ because all elements in z^R_2 are the output of ReLU so they are non-negative, (4) is then split into several terms:

$$\bar{z}_{2,j} = z^R_{2,j} - z^M_{2,j} \tag{5a}$$

$$z'_{2,j} = \max(\bar{z}_{2,j}, 0) \tag{5b}$$

$$z^M_{2,j+1} = z'_{2,j} + z^M_{2,j} \tag{5c}$$

After decomposition using (5), the second term is also a ReLU function so convex relaxation is used, assuming knowledge of the upper-lower bounds of $\bar{z}_{2,j}$. The ReLU and maxpool functions in (3) are then replaced by convex outer bounds linear inequalitiy constraints to form a convex optimization problem. The dual problem can then be written in the form of a dual network (see Supplementary Material A.3.2 for detailed derivation):

$$\max J_\epsilon(\Theta) \quad \text{s.t.} \quad F(\mathbf{d}, \alpha^R, \alpha^M) \tag{6}$$

which has the form of a leaky-ReLU network. If the dual optimal of the convex relaxation problem is positive, then the system verifies the network as robust and if not, the system does not exclude the possibility that the network can be fooled by some perturbation with l_∞-norm of at most ϵ. The variables α^R and α^M are considered to be additional free variables for the dual network, the choice of which affects the precision of the verification bound that is calculated by the dual network. A strategy that is similar to CAP (Wong & Kolter, 2018) is used to determine α^R and α^M.

2.2.2 Determining the upper-lower bound

The node-wise upper-lower bounds are required for the convex relaxation of ReLU functions. To determine the upper-lower bounds, namely $\hat{l}_{2,j} \le \hat{z}_{2,j} \le \hat{u}_{2,j}$, for the input nodes of the ReLU function, a verification problem is formulated that corresponds to the network up to the linear layer before the first ReLU. The node-wise bounds for $\hat{z}_{2,j}$ are determined by evaluating the resulting (smaller) dual network with one-hot input vector $\boldsymbol{e}_j$ (instead of $\mathbf{d}$) (Wong & Kolter, 2018). In terms of the element-wise lower and upper bounds, namely $\bar{l}_{2,j} \le \bar{z}_{2,j} \le \bar{u}_{2,j}$, that pertain to the maxpool functions, each maxpool function is decomposed into multiple ReLU activations (cf. (4)). Thus computing these bounds layer-by-layer as in Wong & Kolter (2018) would be very costly. The values for $\bar{l}_{2,j}$ and $\bar{u}_{2,j}$ can be calculated more efficiently as follows: For (cf. (5a))

$$\bar{z}_{2,j} = z^R_{2,j} - z^M_{2,j},$$

if the element-wise lower and upper bounds for $z^R_{2,j}$ and $z^M_{2,j}$, namely

$$l^R_{2,j} \le z^R_{2,j} \le u^R_{2,j} \quad \text{and} \quad l^M_{2,j} \le z^M_{2,j} \le u^M_{2,j}$$

are known, then the element-wise lower and upper bounds $\bar{l}_{2,j}$ and $\bar{u}_{2,j}$ for $\bar{z}$ are derived as

$$\bar{l}_{2,j} = l^R_{2,j} - u^M_{2,j} \quad \text{and} \quad \bar{u}_{2,j} = u^R_{2,j} - l^M_{2,j}.$$

The elementwise bounds $l^R_{2,j}$ and $u^R_{2,j}$ on the pre-maxpool activations can be computed in a way similar to how the elementwise bounds for pre-ReLU activations are computed in Wong & Kolter (2018). To compute $l^M_{2,j+1}$ and $u^M_{2,j+1}$, recall that

$$z^M_{2,j+1} = \max\left\{z^R_{2,j'} : j' \in [\![0, j]\!]\right\}.$$

Thus, one can take

$$l^M_{2,j+1} = \max\left\{l^R_{2,j'} : j' \in [\![0, j]\!]\right\}, \quad \text{and} \quad u^M_{2,j+1} = \max\left\{u^R_{2,j'} : j' \in [\![0, j]\!]\right\}.$$

Since our algorithm design is primarily an extension of the method proposed by Wong & Kolter (2018), we can deduce the time complexity of our algorithm to be $O(W^2NK)$ by analyzing that the time complexity of Wong & Kolter (2018)'s algorithm is $O(W^2N)$.

Table 1: Network parameters

Dataset	Model	# Hidden layers	# Parameters
MNIST	convSmall	3	89606
	convMed	3	160070
	convBig	7	893418
CIFAR10	convSmall	3	125318
	convMed	3	208198
	convBig	7	2466858
MNIST	conv_S	3	9538
	conv_M	4	19162
	conv_L	4	568426

Table 2: Comparison with previous studies using the MNIST dataset

Model	Training	Accuracy	ϵ	Our Ver	DeepZ Ver	DeepPoly Ver	PRIMA Ver
convSmall	Normal	100	0.03	65	0	40	**79**
convSmall	Fast	100	0.03	85	2	72	**92**
convSmall	PGD	100	0.03	**97**	10	91	**97**
convSmall	PGD	100	0.04	84	0	53	**85**
convMed	Normal	100	0.01	93	89	95	**96**
convMed	Fast	100	0.03	88	49	88	**96**
convMed	PGD	100	0.03	96	82	96	**97**
convMed	PGD	100	0.04	89	15	87	**93**
convBig	Normal	100	0.01	73	0	86	**86**
Model	Training	Accuracy	ϵ	Our Time	DeepZ Time	DeepPoly Time	PRIMA Time
convSmall	Normal	100	0.03	**1.14**	3.76	9.47	229.8
convSmall	Fast	100	0.03	**1.17**	4.55	8.95	71.28
convSmall	PGD	100	0.03	**1.23**	3.68	6.82	25.35
convSmall	PGD	100	0.04	**1.95**	3.90	8.19	149.7
convMed	Normal	100	0.01	**4.67**	5.94	10.74	15.39
convMed	Fast	100	0.03	**4.24**	5.60	10.11	35.16
convMed	PGD	100	0.03	**4.72**	4.73	9.20	26.98
convMed	PGD	100	0.04	**4.56**	5.88	10.93	58.77
convBig	Normal	100	0.01	338	**120.1**	1493	5560

Table 3: Comparison with α, β-CROWN using the MNIST dataset

Model	Training	Accuracy	ϵ	Our Ver	α-β-crown Ver
conv_S	Normal	98	0.03	66	0
conv_S	Fast	99	0.03	94	33
conv_S	PGD	99	0.03	95	37
conv_S	PGD	99	0.04	93	11
conv_M	Normal	99	0.01	97	13
conv_M	Fast	98	0.03	95	13
conv_M	PGD	100	0.03	96	7
conv_M	PGD	100	0.04	90	2
conv_L	Normal	99	0.01	95	0

3 EXPERIMENTS

Our experiments compare CAPM with other neural network verification methods, including DeepZ, DeepPoly, PRIMA, and α, β-CROWN, under various ℓ_∞-norm perturbation budgets and adversarial attacks (FGSM, PGD). Experiments use CNN architectures (convSmall, convMed, convBig, etc. as shown in Table 1) trained on MNIST and CIFAR10, both normally and with adversarial training (Fast and PGD). Verified robustness (abbreviated as Ver.) is defined as the fraction of correctly classified images proven robust against adversarial perturbations, while average verification time (abbreviated as Time) gauges computational cost. The details of experiment can be found in Sec. A.1

The results of MNIST are shown in Table 2 and 3. Based on these tables, CAPM achieves comparable or superior verified robustness to state-of-the-art methods at significantly lower runtime. Although PRIMA achieves the best verified robustness, however, its computational cost is much larger than CAPM and grows with larger perturbation budgets ϵ. On the other hand, CAPM's runtime remains nearly constant across all ϵ. α, β-CROWN faces memory and speed issues on maxpool-based CNNs, especially when it must resort to branch-and-bound (BaB). More experiment result, including the result of CIFAR, can be found in Sec. A.1.2. Overall, CAPM emerges as a robust and efficient solution, particularly well-suited to large-scale verification tasks.

4 CONCLUSION

This study extends Wong & Kolter (2018)'s work to general purpose CNNs with maxpool, padding, and striding operations. The key idea for handing the maxpool function is to decompose it into multiple ReLU functions, while special care is taken to speed-up the computation of element-wise bounds required for the convex relaxation of intermediate ReLUs in maxpool layer. General purpose CNNs are expressed using a dual network, which allows efficient computation of verified bounds for CNNs.

The experimental results show that CAPM outperforms previous methods (DeepZ, DeepPoly, and PRIMA) in terms of verified robustness and computational cost for most adversary budget settings, and especially for large-scale CNNs for color images. For an adversary budget $\epsilon = 0.0024$, the verified robustness for DeepZ, DeepPoly and PRIMA for convSmall CIFAR10 decreases by to 1%, 25%, and 26%, respectively, but CAPM has a verified robustness of 87.5%; For an adversary budget $\epsilon = 0.0006$ for convBig CIFAR10, CAPM is 40-times and 20-times faster than PRIMA and DeepPoly, respectively, and gives a significantly higher verified robustness (see Fig. 3 and Fig. 4).

The proposed method gives comparable or better verification with significantly less runtime cost. Unlike many verification methods, for which runtime increases with the adversary budget ϵ, CAPM has a constant runtime, regardless of the adversary budget, so it can be used for larger-scale CNNs which are usually computationally prohibitive for other verification methods. The proposed verification method is suited for use with large scale CNNs, which are an important element of machine learning services.

This study does provide a more precise and efficient verification for maxpool-based CNNs but the verified network is limited to a specific architecture that is defined in Supplementary Material A.2.1. Future study will involve the design of a verification framework that is applicable to neural networks with a more flexible architecture. Additionally, the method of simplifying certain layers into multiple ReLU layers may likely be limited to maxpool layers only. Therefore, our preliminary future direction will focus on achieving greater flexibility in maxpool-based CNNs. Subsequently, we will continue exploring more flexible neural network architectures, such as residual connections.

ACKNOWLEDGMENTS

This work was supported in part by the Asian Office of Aerospace Research & Development (AOARD) under Grant NTU-112HT911020, National Science and Technology Council of Taiwan under Grant NSTC-112-2221-E-002-204- and NSTC-112-2622-8-002-022 and NSTC-113-2221-E-002-208-, Ministry of Education (MOE) of Taiwan under Grant NTU-113L891406, Ministry of Environment under Grant NTU-113BT911001, and ASUS under Project 112CB244.

REFERENCES

Naveed Akhtar, Jian Liu, and Ajmal Mian. Defense against universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3389–3398, 2018.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. In *Conference on Uncertainty in Artificial Intelligence*, pp. 370–379. PMLR, 2020.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. IEEE, 2017.

Akshay Chaturvedi and Utpal Garain. Mimic and fool: A task-agnostic adversarial attack. *IEEE transactions on neural networks and learning systems*, 32(4):1801–1808, 2020.

Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Cvxpy. Cvxpy/cvxpy: A python-embedded modeling language for convex optimization problems. URL `https://github.com/cvxpy/cvxpy`.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip HS Torr, and M Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021.

Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9185–9193, 2018.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, pp. 3, 2018.

Eth-Sri. Eth-sri/eran: Eth robustness analyzer for deep neural networks. URL `https://github.com/eth-sri/eran`.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 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.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Keji Han, Yuxuan Bai, and Yun Li. A way to explore the lower bound of adversarial perturbation. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 338–341. IEEE, 2020.

Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 142–150, 2015.

Xiaojun Jia, Xingxing Wei, Xiaochun Cao, and Hassan Foroosh. Comdefend: An efficient image compression model to defend adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6084–6092, 2019.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5419–5427, 2018.

Mostafa Mehdipour Ghazi and Hazim Kemal Ekenel. A comprehensive analysis of deep learning based representation for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 34–41, 2016.

Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pp. 3578–3586. PMLR, 2018.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 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.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin T Vechev. Prima: general and precise neural network certification via scalable convex hull approximations. *Proc. ACM Program. Lang.*, 6(POPL):1–33, 2022.

Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4422–4431, 2018.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

Qing Rao and Jelena Frtunikj. Deep learning for self-driving cars: Chances and challenges. In *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, pp. 35–38, 2018.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks, 2020.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32:3353–3364, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T Vechev. Fast and effective robustness certification. *NeurIPS*, 1(4):6, 2018.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019a.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T Vechev. Boosting robustness certification of neural networks. In *ICLR (Poster)*, 2019b.

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems*, 33:1633–1645, 2020.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and 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, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29909–29921. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/fac7fead96dafceaf80c1daffeae82a4-Paper.pdf.

Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5286–5295. PMLR, 2018.

Eric Wong, Frank R Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. *Advances in Neural Information Processing Systems*, 31:8400–8409, 2018.

Eric Wong, Frank Schmidt, and Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. In *International Conference on Machine Learning*, pp. 6808–6817. PMLR, 2019.

Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020.

Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 501–509, 2019.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *CoRR*, abs/2011.13824, 2020. URL https://arxiv.org/abs/2011.13824.

Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *arXiv preprint arXiv:1906.06316*, 2019.

A APPENDIX

A.1 EXPERIMENT DETAILS

This section determines the verified robustness and the average verification time for CAPM, DeepZ, DeepPoly, PRIMA (Müller et al., 2022) and α,β-CROWN for a l_∞ norm-bounded perturbation of various budgets and for various attack schemes, such as FGSM and PGD. Sec. A.1.1 details the experimental network architecture and the input dataset. Sec. A.1.2 compares the results for this study with those of previous studies (DeepZ, DeepPoly, PRIMA and α,β-CROWN) in terms of the verified robustness and the average verification time, and Supplementary Material A.6 illustrates how we reproduced the state-of-the-art methods so that we can have a fair comparison with them. Experiments in Supplementary Materials A.4.2 also demonstrate that the neural network verification problem cannot be simply evaluated using a Monte-Carlo simulation. All experiments were conducted on a 2.6 GHz 14 core Intel(R) Xeon(R) CPU E5-2690 v4 with a 512 GB main memory.

A.1.1 EXPERIMENT SETTING

Benchmarks: Robustness is calculated against adversarial examples on several networks that are trained using different methods:

- **Dataset:** Models are trained using the MNIST and CIFAR10 datasets. Images are normalized using the default setting for DeepPoly (Singh et al., 2019a). For MNIST, the mean and standard deviation is 0.5 and 0.5, respectively. For CIFAR 10, the mean and standard deviation of the RGB channels is $(0.485, 0.456, 0.406)$ and $(0.229, 0.224, 0.225)$, respectively.
- **Architecture of neural networks:** There are no empirical verificaion results on maxpool-based CNNs so maxpool layers are added to the common benchmark networks, convSmall, convMed and convBig in Mirman et al. (2018). The parameter for striding and padding is adjusted to achieve a similar number of parameters to previous studies. Information about these 6 networks is shown in Table 1. The detail structures are shown in Supplementary Material A.5. Moreover, although the authors of α,β-CROWN didn't conduct experiments in maxpool-based CNNs before and they did make some additional assumptions on their maxpool layers, we would like to compare with them in maxpool-based CNNs. Hence, we also create the network benchmark conv_S, conv_M and conv_L for their settings.
- **Training methods:** We compared the verification results of CNNs trained either normally (without adversarial training) or with adversarial examples such as Fast-Adversarial (Wong et al., 2020) and PGD (Madry et al., 2018).
- **Performance metrics:** The performance of neural network verification is often evaluated through the following metrics (Singh et al., 2019a):
  - *Verified robustness:* This is expressed as the number of images verified to be resilient to adversary example attack, divided by the total number of accurate images. This ratio represents the analysis precision of a verifier when a neural network is applied to a test image dataset that is subject to attack by an adversarial example.
  - *Average verified time:* This is the total time that is required by the verification algorithm to verify images, divided by the total number of images.

Robustness evaluation: For each test dataset, the settings for DeepPoly (Singh et al., 2019a) are used and the top 100 clean images are used as the evaluation test dataset. Adversarial examples are generated by adding to clean images with l_∞ norm-bounded perturbation for various budgets ϵ and various attack schemes, such as FGSM and PGD. The generated adversarial examples are then applied to the neural network to compare the accuracy of lower bounds that are evaluated using various verification methods. A better verification method must give a tighter (higher) accuracy for the lower bound and never exceeds that of existing attack schemes. The implementation details for DeepPoly, PRIMA and α,β-CROWN are described as follows:

- **DeepZono and DeepPoly**: We follow the implementation as suggested by the default command in their GitHub (Eth-Sri).

- **PRIMA**: We use exactly the same configuration mentioned in Müller et al. (2022) for convSmall and convBig. Since PRIMA didn't report any results on convMed, and that convMed has the same number of layers as convSmall, we use the same configuration that PRIMA applied to convSmall on convMed as well.
- **α,β-CROWN**: We used the default parameters and set the parameter conv mode to matrix in order to enable the operation of the BaB algorithm. The examples that would trigger the BaB algorithm were considered not verified. This adjustment was made because the implementation of the BaB algorithm on maxpool-based CNNs results in excessive GPU memory requirements (more than 13GB) and extended execution times (more than 5 minutes per example).

A.1.2 EXPERIMENTAL RESULTS

The results for CAPM are compared with those of previous studies (DeepZ, DeepPoly, PRIMA) in terms of the verified robustness and average runtime metrics for various adversary budgets ϵ for six different networks, as shown in Fig. 3 and Fig. 4. The classification accuracy [1] for a real-world PGD attack (orange solid line with triangle marker) is also determined. A verification method must demonstrate verified robustness that is no greater than the accuracy of real-world attack schemes. Fig. 3 and Fig. 4 illustrate that CAPM achieves better verification robustness than all other schemes for all combination of the CIFAR10 dataset and has a much lower computational cost. The computational cost of CAPM is independent of ϵ, because a different adversary budget corresponds to the same dual network architecture so computational costs are similar. However, the verification time that is required by PRIMA increases as ϵ increases, possibly because as ϵ increases, the intermediate adversarial polytope becomes more complicated and must be described by a more complex MILP optimization problem. This demonstrates the promising potential of CAPM towards verification of large scale CNNs and colour images. Due to the excessive GPU memory requirements of α,β-CROWN, we did not include α,β-CROWN in this set of experiments.

The results for the MNIST dataset are shown in Table 2 and Table 3. PRIMA and CAPM both yield a higher (tighter) verification robustness than DeepPoly or DeepZ (cf. Fig. 2). CAPM also has a comparable verification robustness to PRIMA, but average runtime is significantly reduced. For larger networks such as convBig, only CAPM achieves effective verified robustness in a reasonable runtime.

For α,β-CROWN, the BaB algorithm they employ is not suitable for maxpool CNNs, leading to excessive GPU memory requirements (more than 13GB) and prolonged execution times (more than 5 minutes per example). Therefore, in this experiment, we considered examples that would trigger the BaB algorithm as not verified. Consequently, the performance of α,β-CROWN is not satisfactory in our study.

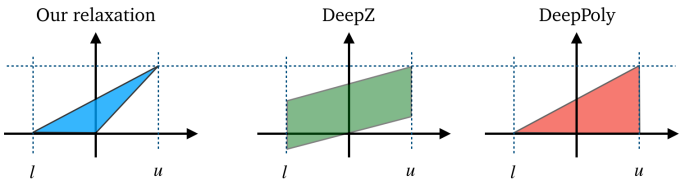

Figure 2: The difference in convex relaxation for the method of this study, DeepZ, and DeepPoly.

CAPM has the following advantages, compared to other state of the arts verification methods (DeepZ, DeepPoly, and PRIMA):

- CAPM achieves comparable or better verification robustness with significantly less runtime cost.

[1] Here we follow the settings in Singh et al. (2019a); Müller et al. (2022) and only test on images which were correctly classified before any perturbation is added.

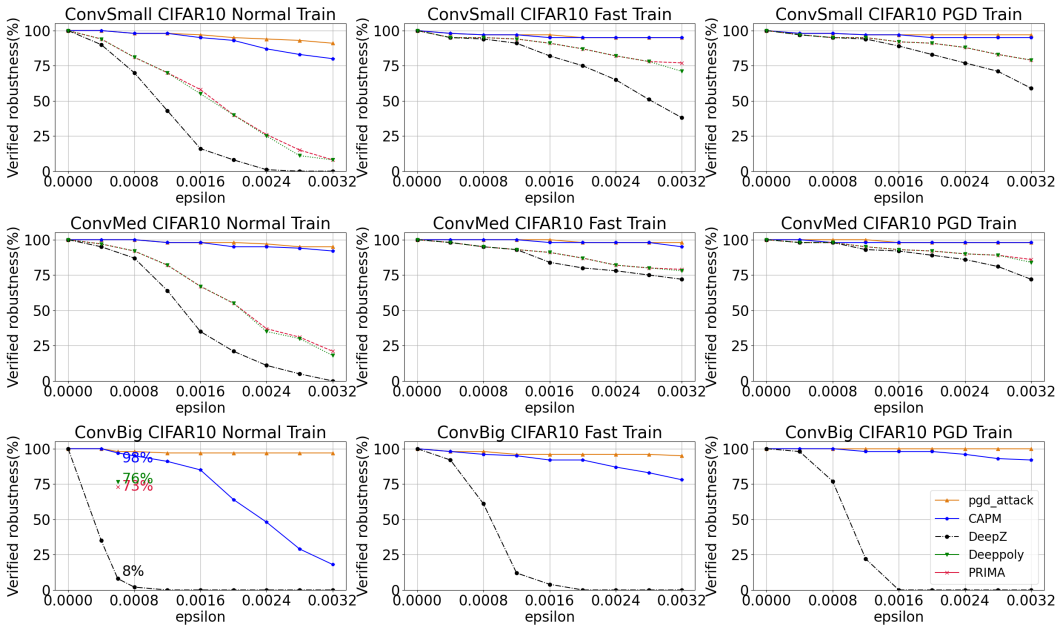

Figure 3: Verified robustness for ϵ perturbations under l_∞-norm by CAPM (black solid line with pentagon marker), DeepPoly (green dotted line with triangle marker), DeepZ (black dashdot line with circle marker), and PRIMA (red dashed line with x marker) for convSmall CIFAR10, convMed CIFAR10, and convBig CIFAR10. The orange solid line with triangle markers is the classification accuracy for a PGD attack. PRIMA and DeepPoly are both prohibitively computationally costly for convBig CIFAR10, so we only show the result for DeepZ.

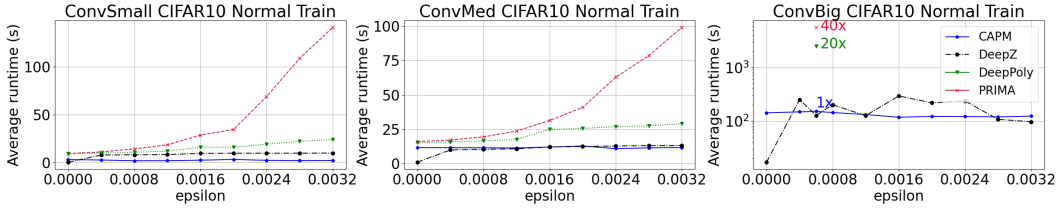

Figure 4: Runtime for ϵ perturbations under l_∞-norm for CAPM (black solid line with pentagon marker), DeepPoly (green dotted line with triangle marker), DeepZ (black dashdot line with circle marker) and PRIMA (red dashed line with x marker) for convSmall CIFAR10, convMed CIFAR10 and convBig CIFAR10.

- Unlike other verification methods for which runtime increases with the adversary budget ϵ, CAPM has a constant runtime that is independent of ϵ. For larger networks such as convBig, only CAPM demonstrates a feasible runtime and good verified robustness.

- CAPM is especially suitable for larger-scale maxpool-based CNNs that are designed for color images. Using the CIFAR10 dataset, CAPM achieves a significantly tighter verified robustness at a much lower computational cost. And due to computational costs, we did not continue our experiments on larger and more general networks. Nevertheless, from Tables 2 and 3 in our experiments, it can be observed that our CAPM not only exhibits a lower growth rate in computation time as the neural network scales up but also maintains nearly equal accuracy under significantly lower computation time compared to PRIMA.

A.2 MAXPOOL-BASED CNN ARCHITECTURE

As mentioned in section 2.1, the network f in consideration is a maxpool-based CNN, which consists of k convolution layers with ReLU activation and maxpool functions, followed by flattening and t fully connected layers with ReLU activation. In this manuscript we separate layers $l-1$ and l with the linear operations. That is, the first layer simply contains a convolution operation; and from layer 2 to $k-1$, there are ReLU, maxpool, and convolution operations in order; while the k-th layer contains ReLU, maxpool, flatten, and linear (fully connected) operations. After the flatten operation is the fully-connected part of the network. Denote $T = k + t$ as the total number of layers in f, then within layers $k+1$ to T there are ReLU and linear (fully connected) operations in order, while the last layer is a pure output layer containing no operations. In the following derivation, we consider CNN with $k \geq 2$ and $t \geq 1$. The explicit notations pertaining to all intermediate results during the calculation of f are specified in Sec. A.2.1 and Table 4.

Table 4: Basic notations, where $[\![m, n]\!]$ denotes the set of integers from m to n.

layer	notation	description
$l = 1$	$\mathbf{z}_l$	the (perturbed) input image.
$l \in [\![1, k]\!]$	c_l	the channel number of the l-th layer feature maps.
	N_l	the width and height of feature map $\mathbf{z}_l$.
$l \in [\![1, k-1]\!]$	k_l^{cv}	the size of the l-th convolution kernel.
	s_l^{cv}	the stride of the l-th convolution kernel.
	p_l^{cv}	the padding of the l-th convolution kernel (choose 0 or greater).
$l \in [\![2, k]\!]$	$\hat{\mathbf{z}}_l$	the feature map after convolution which will be fed to ReLU.
	$\mathbf{z}_l^R$	the feature map after ReLU which will be fed to maxpool.
	$\mathbf{z}_l$	the feature map after maxpool which will be fed to convolution.
	k_l^M	the size of the l-th maxpool kernel.
	s_l^M	the size of the l-th maxpool stride.
	q_l	the width and height of feature maps $\hat{\boldsymbol{z}}_l$ and $\boldsymbol{z}_l^R$.
$l \in [\![k+1, k+t]\!]$	$\hat{\mathbf{z}}_l'$	the feature vector after (fully connected) linear operator in layer $l-1$, which will be fed to ReLU.
$l \in [\![k, k+t-1]\!]$	$\tilde{\mathbf{z}}_l$	the feature vector before (fully connected) linear operator in layer l.
$l \in [\![k, k+t]\!]$	a_l	the length of vectors $\hat{\mathbf{z}}_l'$ and $\tilde{\mathbf{z}}_l$.

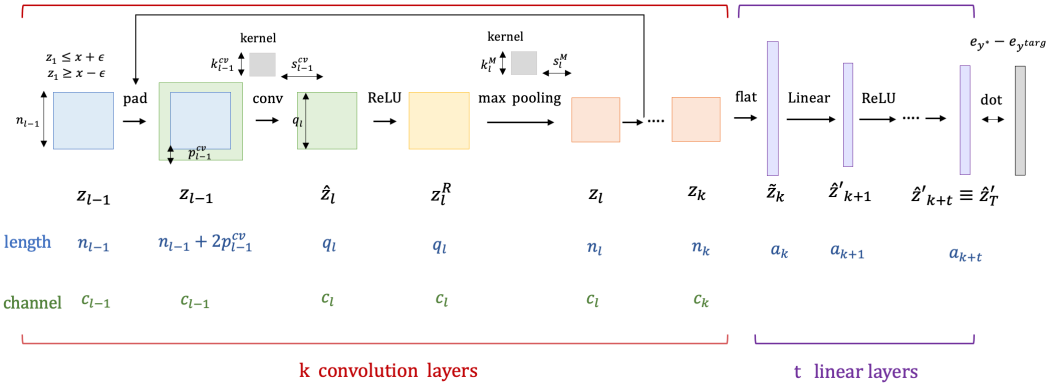

Figure 5: Data flow and notations.

A.2.1 NOTATION AND DATA FLOW

We use the tuple (l, c, m, n) to indicate the pixel located at the m-th row and n-th column in the c-th channel of the corresponding l-th layer feature map. For instance, such a pixel in the feature map $\mathbf{z}_l$ is indicated as $z_{l,c,m,n}$.

Table 5: Fixed region notations.

layer	notation description
$l = 1, ..., k$	$S_l^{IN} = [\![0, c_l - 1]\!] \times [\![0, N_l - 1]\!] \times [\![0, N_l - 1]\!]$
$l = 1, ..., k-1$	$S_l^{UD} = [\![0, c_l - 1]\!] \times ([\![-p_l^{cv}, -1]\!] \cup [\![N_l, N_l + p_l^{cv} - 1]\!]) \times [\![-p_l^{cv}, N_l + p_l^{cv} - 1]\!]$ $S_l^{LR} = [\![0, c_l - 1]\!] \times [\![0, N_l - 1]\!] \times ([\![-p_l^{cv}, -1]\!] \cup [\![N_l, N_l + p_l^{cv} - 1]\!])$ $S_l = S_l^{IN} \cup S_l^{UD} \cup S_l^{LR}$
$l = 2, ..., k$	$Q_l^{IN} = [\![0, c_l - 1]\!] \times [\![0, q_l - 1]\!] \times [\![0, q_l - 1]\!]$

To consider padding which is a common practice in convolutions, we extend the pixel region of the feature map $\mathbf{z}_l$, which is the input to the convolution operation, to $S_l = S_l^{IN} \cup S_l^{UD} \cup S_l^{LR}$ as indicated in Table 5 and Fig. 6. Here S_l^{IN} represents the index region of the feature map $\mathbf{z}_l$ without padding, while S_l^{UD} represents the padded index region located at the upper/down-sides of the feature map $\mathbf{z}_l$, and S_l^{LR} represents the padded index region located at the left/right-sides of the feature map $\mathbf{z}_l$.

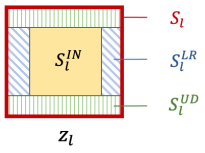

Figure 6: Pixel regions of feature map $\mathbf{z}_l$.

Refer to Fig. 5, in the convolution part of the network, which corresponds to layers $l \in [\![1, k]\!]$, we first compute feature map $\hat{\mathbf{z}}_l$ as the convolution of $\mathbf{z}_{l-1}$. Starting from $\hat{\mathbf{z}}_l$, we consecutively apply ReLU and maxpool to compute intermediate results $\mathbf{z}_l^R$ and $\mathbf{z}_l$, respectively. Note that $\hat{\mathbf{z}}_l$ and $\mathbf{z}_l^R$ both have the same index region Q_l^{IN} (cf. Table 5) as ReLU does not change the size of feature map. The aforementioned process can be iterated until reaching $\mathbf{z}_k$, to which the flat kernel is applied to flatten the feature map $\mathbf{z}_k$ into vector form $\tilde{\mathbf{z}}_k$. In the fully-connected part of the network, which corresponds to layers $l \in [\![k+1, k+t]\!]$, we first compute feature vector $\hat{\mathbf{z}}_l'$ as the result of the fully connected linear operation applied on $\tilde{\mathbf{z}}_{l-1}$, followed by ReLU to get $\tilde{\mathbf{z}}_l$. We iterate the aforementioned process until finally reaching $\hat{\mathbf{z}}_T'$, which is the output of network f.

Before introducing the whole process of network f, we first introduce notations for the convolution kernel and the flat kernel as below:

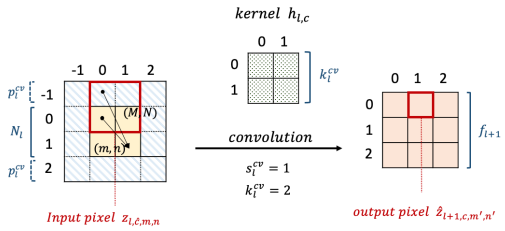

Figure 7: After convolution the output pixel $\hat{z}_{l+1,c,m',n'}$ is the sum of products between $z_{l,\hat{c},m,n}$ and $h_{l,c}(\hat{c}, M, N)$, where $M = m + p_l^{cv} - s_l^{cv} m'$ and $N = n + p_l^{cv} - s_l^{cv} n'$.

- Convolution kernel $h_{l,c}$: As illustrated in Fig. 7, we use $h_{l,c}$ to represent the c-th convolution kernel in the l-th layer, where $l \in [\![1, k-1]\!]$ and $c \in [\![0, c_{l+1} - 1]\!]$. In addition, each kernel contains c_l channels with both height and width k_l^{cv}, namely the index region of $h_{l,c}$ is $[\![0, c_l - 1]\!] \times [\![0, k_l^{cv} - 1]\!] \times [\![0, k_l^{cv} - 1]\!]$. Here we use $h_{l,c}(\hat{c}, \hat{m}, \hat{n})$ to refer to the kernel value at the $\hat{c}$-th channel, $\hat{m}$-th row and $\hat{n}$-th column, and define $h_{l,c}(\hat{c}, \hat{m}, \hat{n}) = 0$ for each index $(\hat{c}, \hat{m}, \hat{n})$ that is out of the range $[\![0, c_l - 1]\!] \times [\![0, k_l^{cv} - 1]\!] \times [\![0, k_l^{cv} - 1]\!]$.

- Flat kernel $\hat{\boldsymbol{W}}$: The flat kernel unfolds the feature map $\boldsymbol{z}_k$ to the vector $\tilde{\boldsymbol{z}}_k$ in the flattening layer, namely

$$\tilde{z}_{k,a} = \sum_{(c,m,n)\in S_k^{IN}} (\hat{W}_{c,m,n})_a z_{k,c,m,n}, \quad a \in [\![0, a_k - 1]\!],$$

where $(\hat{W}_{c,m,n})_a = \mathbb{1}\{a = c_k N_k N_k + m N_k + n\}$ for $(c,m,n) \in S_k^{IN}$, and that $a_k = c_k N_k N_k$.

Before introducing the details of the math operations processing through network f, we first illustrate how to decompose the maxpool: Instead of considering the whole group of candidates as the input of maxpool, e.g., $y_{out} = \max(x_1, x_2, ..., x_N)$, we split the maxpool function into several lines of one-by-one comparisons, each considering only one additional input

$$\begin{aligned} y_1 &= \max(x_1, 0) = \text{ReLU}(x_1 - 0) + 0, \\ y_2 &= \max(x_2, y_1) = \text{ReLU}(x_2 - y_1) + y_1, \\ &\vdots \\ y_{out} &= \max(x_N, y_{N-1}) = \text{ReLU}(x_N - y_{N-1}) + y_{N-1}. \end{aligned} \tag{7}$$

Note that we assume all of the inputs $x_1, ..., x_N$ are non-negative, since they are drawn from the output of ReLU. The operations through network f is given as follows:

Starting with feature map $\boldsymbol{z}_{l-1}$ ($l \in [\![2, k]\!]$) with index region S_{l-1}^{IN}, to apply convolution with zero-padding, we first pad the four boundaries of $\boldsymbol{z}_{l-1}$ with 0, namely

$$z_{l-1,c,m,n} = 0, \ (c,m,n) \in S_{l-1}^{UD} \cup S_{l-1}^{LR}, \tag{8}$$

$$\hat{z}_{l,c,m',n'} = b_{l-1,c} + \sum_{(\hat{c},m,n)\in S_{l-1}} h_{l-1,c}(\hat{c}, m + p_{l-1}^{cv} - s_{l-1}^{cv} m', n + p_{l-1}^{cv} - s_{l-1}^{cv} n') z_{l-1,\hat{c},m,n}, (c,m',n') \in Q_l^{IN} \sim l \in [\![2,k]\!]. \tag{9}$$

By taking convolution between the convolution kernel $h_{l-1,c}$ and the padded $\boldsymbol{z}_{l-1}$, we compute $\hat{\boldsymbol{z}}_l$ as in (9). As such, $\boldsymbol{z}_l^R$ is obtained by applying ReLU to $\hat{\boldsymbol{z}}_l$ as follows:

$$z_{l,c,m,n}^R = \max\left\{\hat{z}_{l,c,m,n}, 0\right\}, \ (c,m,n) \in Q_l^{IN}. \tag{10}$$

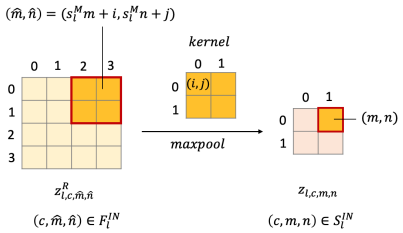

Figure 8: This example demonstrates the relationship between the output pixel $z_{l,c,m,n}$ and input pixels $z^R_{l,c,\hat{m},\hat{n}}$ when applying maxpool function. The input map z_l^R has 1 channel with width and height both equal to 4 ($q_l = 4$), and the kernel size is 2×2. As such, the pixel $(0,0,1)$ in the output map is the maximum of the input pixels $(0,0,2), (0,0,3), (0,1,2), (0,1,3)$.

Regarding the maxpool layer, as illustrated in Fig. 8, one has

$$z_{l,c,m,n} = \max_{\substack{i\in[\![0,k_l^M-1]\!],\\ j\in[\![0,k_l^M-1]\!]}} \left\{ z^R_{l,c,s_l^M m+i, s_l^M n+j} \right\} \tag{11}$$

where k_l^M and s_l^M indicate the kernel size and stride, respectively. By applying the decomposition trick to maxpool as in (7), one may rewrite (11) as $z_{l,c,m,n} = (z^M_{l,c,m,n})_{k_l^M k_l^M}$ for $(c,m,n) \in S_l^{IN}$, where we denote $(z^M_{l,c,m,n})_0 = 0$ and

$$\begin{aligned}
&(z^M_{l,c,m,n})_{ik_l^M+j+1} \\
&= \max\left\{ z^R_{l,c,s_l^M m+i',s_l^M n+j'} \ : \ \begin{smallmatrix} i'\in[\![0,k_l^M-1]\!], \\ j'\in[\![0,k_l^M-1]\!], \\ i'k_l^M+j'\le ik_l^M+j \end{smallmatrix} \right\} \\
&= \max\left\{ z^R_{l,c,s_l^M m+i,s_l^M n+j},\ (z^M_{l,c,m,n})_{ik_l^M+j} \right\} \\
&= \max\left\{ z^R_{l,c,s_l^M m+i,s_l^M n+j} - (z^M_{l,c,m,n})_{ik_l^M+j} \right\} + (z^M_{l,c,m,n})_{ik_l^M+j}
\end{aligned} \tag{12}$$

for $i \in [\![0, k_l^M - 1]\!]$ and $j \in [\![0, k_l^M - 1]\!]$, which can be further decomposed as follows:

$$\begin{cases}
(\bar{z}_{l,c,m,n})_{ik_l^M+j} = z^R_{l,c,s_l^M m+i,s_l^M n+j} - (z^M_{l,c,m,n})_{ik_l^M+j} \\
(z'_{l,c,m,n})_{ik_l^M+j} = \quad \max \quad \{(\bar{z}_{l,c,m,n})_{ik_l^M+j}, 0\} \\
(z^M_{l,c,m,n})_{ik_l^M+j+1} = (z'_{l,c,m,n})_{ik_l^M+j} + (z^M_{l,c,m,n})_{ik_l^M+j}
\end{cases}. \tag{13}$$

Once the convolution part of the neural network is completed, we get z_k and convert it into a vector $\tilde{z}_k$ by flat kernel $\hat{\boldsymbol{W}}$, namely,

$$\tilde{z}_{k,a} = \sum_{(c,m,n)\in S_k^{IN}} (\hat{W}_{c,m,n})_a z_{k,c,m,n} \tag{14}$$

for all $a \in [\![0, a_k - 1]\!]$. Finally, $\tilde{z}_k$ is fed into linear part, where the linear and ReLU operations can be written as:

$$\hat{\boldsymbol{z}}'_{l+1} = \boldsymbol{W}_l \tilde{\boldsymbol{z}}_l + \boldsymbol{b}_l,\ l \in [\![k, k+t-1]\!] \tag{15a}$$

$$\tilde{\boldsymbol{z}}_l = \quad \max(\hat{\boldsymbol{z}}'_l, 0),\ l \in [\![k+1, k+t-1]\!]. \tag{15b}$$

A.2.2 Convex outer bound

In order to make the feasible solution set a convex set, we construct the convex outer bound for the activation function. First, we consider the ReLU functions in network f, listed in the following:

$$z^R_{l,c,m,n} = \max(\hat{z}_{l,c,m,n}, 0),\ l \in [\![2,k]\!], (c,m,n) \in Q_l^{IN} \tag{16a}$$

$$\begin{aligned}
&(z'_{l,c,m,n})_{ik_l^M+j} = \max((\bar{z}_{l,c,m,n})_{ik_l^M+j}, 0), \\
&l \in [\![2,k]\!], (c,m,n) \in S_l^{IN}, (i,j) \in [\![0, k_l^M-1]\!] \times [\![0, k_l^M-1]\!]
\end{aligned} \tag{16b}$$

$$\tilde{\boldsymbol{z}}_l = \max(\hat{\boldsymbol{z}}'_l, \boldsymbol{0}),\ l \in [\![k+1, k+t-1]\!] \tag{16c}$$

where $\hat{z}_{l,c,m,n}$, $(\bar{z}_{l,c,m,n})_{ik_l^M+j}$, and $\hat{\boldsymbol{z}}'_l$ are given by (9), (13) and (15a). Suppose for now that we have already known the upper and lower bounds for each entity in the right hand side of (16) :

$$\hat{l}_{l,c,m,n} < \hat{z}_{l,c,m,n} < \hat{u}_{l,c,m,n} \tag{17a}$$

$$(\bar{l}_{l,c,m,n})_{ik_l^M+j} < (\bar{z}_{l,c,m,n})_{ik_l^M+j} < (\bar{u}_{l,c,m,n})_{ik_l^M+j} \tag{17b}$$

$$l'_{l,a} < \hat{z}'_{l,a} < u'_{l,a} \tag{17c}$$

Following the idea of (Wong & Kolter, 2018), we categorize the variables $\{\hat{z}_{l,c,m,n}\}$ into three clusters according to the sign of their upper-lower bounds, namely

$$\begin{aligned}
\mathcal{I}_l^- &= \{(c,m,n) \in Q_l^{IN} \mid \hat{l}_{l,c,m,n} \le \hat{u}_{l,c,m,n} \le 0\} \\
\mathcal{I}_l^+ &= \{(c,m,n) \in Q_l^{IN} \mid 0 < \hat{l}_{l,c,m,n} \le \hat{u}_{l,c,m,n}\} \\
\mathcal{I}_l &= \{(c,m,n) \in Q_l^{IN} \mid \hat{l}_{l,c,m,n} \le 0 < \hat{u}_{l,c,m,n}\}
\end{aligned} \tag{18}$$

As illustrated in Fig. 9, for scenarios $\mathcal{I}_l^-$ and $\mathcal{I}_l^+$, the ReLU function $z^R_{l,c,m,n} = \max(\hat{z}_{l,c,m,n}, 0)$ is essentially zero function ($z^R_{l,c,m,n} = 0$) and identity function ($z^R_{l,c,m,n} = \hat{z}_{l,c,m,n}$), respectively. For

scenario $\mathcal{I}_l$, we construct the convex outer bound (Wong & Kolter, 2018) for the ReLU function as follows:

$$\begin{aligned} &z^R_{l,c,m,n} \geq 0, \\ &z^R_{l,c,m,n} \geq \hat{z}_{l,c,m,n} \\ &(\hat{u}_{l,c,m,n} - \hat{l}_{l,c,m,n}) z^R_{l,c,m,n} \leq \hat{u}_{l,c,m,n} \hat{z}_{l,c,m,n} - \hat{u}_{l,c,m,n} \hat{l}_{l,c,m,n} \end{aligned} \tag{19}$$

The convex outer bounds for ReLU functions pertaining to $\tilde{\boldsymbol{z}}_l$ and $(z'_{l,c,m,n})_{ik_l^M+j}$ are constructed analogously. We cluster the variables $\{\hat{z}'_{l,a}\}$ and $\{(\bar{z}_{l,c,m,n})_a\}$ into three different scenarios accordingly:

$$\begin{aligned} \hat{\mathcal{I}}_l^- &= \{a \in [\![0, a_{l-1}]\!] \mid l'_{l,a} \leq u'_{l,a} \leq 0\} \\ \hat{\mathcal{I}}_l^+ &= \{a \in [\![0, a_{l-1}]\!] \mid 0 < l'_{l,a} \leq u'_{l,a}\} \\ \hat{\mathcal{I}}_l &= \{a \in [\![0, a_{l-1}]\!] \mid l'_{l,a} \leq 0 < u'_{l,a}\} \end{aligned} \tag{20}$$

$$\begin{aligned} \bar{\mathcal{I}}_l^- &= \{(c,m,n,a) \in S_l^{IN} \times [\![0, k_l^M k_l^M - 1]\!] \mid (\bar{l}_{l,c,m,n})_a \leq (\bar{u}_{l,c,m,n})_a \leq 0\} \\ \bar{\mathcal{I}}_l^+ &= \{(c,m,n,a) \in S_l^{IN} \times [\![0, k_l^M k_l^M - 1]\!] \mid 0 < (\bar{l}_{l,c,m,n})_a \leq (\bar{u}_{l,c,m,n})_a\} \\ \bar{\mathcal{I}}_l &= \{(c,m,n,a) \in S_l^{IN} \times [\![0, k_l^M k_l^M - 1]\!] \mid (\bar{l}_{l,c,m,n})_a \leq 0 < (\bar{u}_{l,c,m,n})_a\} \end{aligned} \tag{21}$$

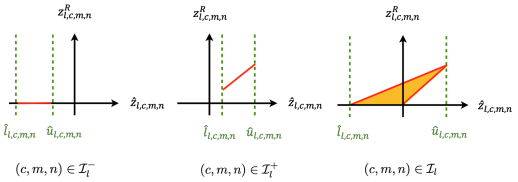

Figure 9: Convex outer bound for ReLU function under various scenarios.

A.3 Formulation of the Lagrangian dual problem

As mentioned earlier, we solve the verification problem with the dual-view approach and take the best dual feasible solution as a lower bound for the verification problem. The primal and dual problems are given in Sec. A.3.1 and A.3.2, where the dual problem is further simplified to a dual network in Sec. A.3.3.

A.3.1 Primal problem

By summarizing Sec. A.2 (cf. (1),(2),(8)-(21)), the verification problem can be formulated as the convex optimization problem.

A.3.2 Dual problem

We associated the dual variables (Boyd et al., 2004) with each of the constraints as follows:

- Norm-bound of the perturbation

$$\begin{aligned} &z_{1,c,m,n} - (x_{c,m,n} + \epsilon_{c,m,n}) \leq 0 \Rightarrow \xi^+_{c,m,n} \in \mathbb{R}_{\geq 0},\ (c,m,n) \in S_1^{IN} \\ &-z_{1,c,m,n} + (x_{c,m,n} - \epsilon_{c,m,n}) \leq 0 \Rightarrow \xi^-_{c,m,n} \in \mathbb{R}_{\geq 0},\ (c,m,n) \in S_1^{IN} \end{aligned}$$

- Padding

$$z_{l,c,m,n} = 0 \Rightarrow \nu^p_{l,c,m,n} \in \mathbb{R},\ (c,m,n) \in S_l^{UD} \cup S_l^{LR},\ l \in [\![1, k-1]\!]$$

- Convolution

$$\begin{aligned} &\hat{z}_{l,c,m',n'} - b_{l-1,c} - \sum_{(\hat{c},m,n) \in S_{l-1}} h_{l-1,c}(\hat{c}, m + p^{cv}_{l-1} - s^{cv}_{l-1} m', n + p^{cv}_{l-1} - s^{cv}_{l-1} n') z_{l-1,\hat{c},m,n} = 0 \\ &\Rightarrow \nu_{l,c,m',n'} \in \mathbb{R}, (c,m',n') \in F_l^{IN},\ l \in [\![2, k]\!] \end{aligned} \tag{23}$$

$$
\begin{aligned}
\text{minimize}\quad & \boldsymbol{d}^T\hat{\mathbf{z}}'_T \equiv (\mathbf{e}_{y^*}-\mathbf{e}_{y^{targ}})^T\hat{\mathbf{z}}'_T \\
\text{subject to}\quad & \mathbf{z}_1-(\mathbf{x}+\boldsymbol{\epsilon})\leq\mathbf{0} \\
& -\mathbf{z}_1+(\mathbf{x}-\boldsymbol{\epsilon})\leq\mathbf{0} \\
& z_{l-1,c,m,n}=0,\ (c,m,n)\in S^{UD}_{l-1}\cup S^{LR}_{l-1},\ l\in[\![2,k]\!] \\
& \hat{z}_{l,c,m',n'}-b_{l-1,c}-\sum_{(\hat{c},m,n)\in S_{l-1}} h_{l-1,c}(\hat{c},m+p^{cv}_{l-1}-s^{cv}_{l-1}m',n+p^{cv}_{l-1}-s^{cv}_{l-1}n')z_{l-1,\hat{c},m,n}=0,\ (c,m',n')\in Q^{IN}_l \\
& \text{, for } l\in[\![2,k]\!] \\
& \begin{cases} \left.\begin{aligned} &-z^R_{l,c,m,n}=0 && ,(c,m,n)\in\mathcal{I}^-_l \\ &\hat{z}_{l,c,m,n}-z^R_{l,c,m,n}=0 && ,(c,m,n)\in\mathcal{I}^+_l \\ &-z^R_{l,c,m,n}\leq 0 \\ &\hat{z}_{l,c,m,n}-z^R_{l,c,m,n}\leq 0 && ,(c,m,n)\in\mathcal{I}_l \\ &(\hat{u}_{l,c,m,n}-\hat{l}_{l,c,m,n})z^R_{l,c,m,n}-\hat{u}_{l,c,m,n}\hat{z}_{l,c,m,n}+\hat{u}_{l,c,m,n}\hat{l}_{l,c,m,n}\leq 0 \end{aligned}\right\}, l\in[\![2,k]\!] \\ \left.\begin{aligned} &(z^M_{l,c,m,n})_0=0 \\ &(\bar{z}_{l,c,m,n})_{ik^M_l+j}-[z^R_{l,c,s^M_l m+i,s^M_l n+j}-(z^M_{l,c,m,n})_{ik^M_l+j}]=0,\ i\in[\![0,k^M_l-1]\!],\ j\in[\![0,k^M_l-1]\!],\ (c,m,n)\in S^{IN}_l \\ &\begin{cases} \left.\begin{aligned} &-(z'_{l,c,m,n})_a=0 && ,(c,m,n,a)\in\bar{\mathcal{I}}^-_l \\ &(\bar{z}_{l,c,m,n})_a-(z'_{l,c,m,n})_a=0 && ,(c,m,n,a)\in\bar{\mathcal{I}}^+_l \\ &-(z'_{l,c,m,n})_a\leq 0 \\ &(\bar{z}_{l,c,m,n})_a-(z'_{l,c,m,n})_a\leq 0 \\ &[(\bar{u}_{l,c,m,n})_a-(\bar{l}_{l,c,m,n})_a](z'_{l,c,m,n})_a-(\bar{u}_{l,c,m,n})_a(\bar{z}_{l,c,m,n})_a+(\bar{u}_{l,c,m,n})_a(\bar{l}_{l,c,m,n})_a\leq 0 \end{aligned}\right\} && ,(c,m,n,a)\in\bar{\mathcal{I}}_l \end{cases} \\ &(z^M_{l,c,m,n})_{a+1}-[(z'_{l,c,m,n})_a+(z^M_{l,c,m,n})_a]=0,\ (c,m,n)\in S^{IN}_l,\ a\in[\![0,k^M_l k^M_l-1]\!] \\ &z_{l,c,m,n}-(z^M_{l,c,m,n})_{k^M_l k^M_l}=0,\ (c,m,n)\in S^{IN}_l \end{aligned}\right\} \\
& \text{, where } l\in[\![2,k]\!] \\
& \tilde{z}_{k,a}-\sum_{(c,m,n)\in S^{IN}_k}(\hat{W}_{c,m,n})_a z_{k,c,m,n}=0,\ a\in[\![0,a_k-1]\!] \\
& \hat{\mathbf{z}}'_{l+1}-(\mathbf{W}_l\tilde{\mathbf{z}}_l+\mathbf{b}_l)=\mathbf{0},\ l\in[\![k,k+t-1]\!] \\
& \left.\begin{cases} \left.\begin{aligned} &-\tilde{z}_{l,a}=0 && ,a\in\hat{\mathcal{I}}^-_l \\ &\hat{z}'_{l,a}-\tilde{z}_{l,a}=0 && ,a\in\hat{\mathcal{I}}^+_l \\ &-\tilde{z}_{l,a}\leq 0 \\ &\hat{z}'_{l,a}-\tilde{z}_{l,a}\leq 0 && ,a\in\hat{\mathcal{I}}_l \\ &(u'_{l,a}-l'_{l,a})\tilde{z}_{l,a}-u'_{l,a}\hat{z}'_{l,a}+u'_{l,a}l'_{l,a}\leq 0 \end{aligned}\right\} \end{cases}\right\},\ l\in[\![k+1,k+t-1]\!] \\
\text{variables}\quad & z_{l,c,m,n},\ (c,m,n)\in S_l,\ l\in[\![1,k-1]\!] \\
& z_{k,c,m,n},\ (c,m,n)\in S^{IN}_k \\
& \hat{z}_{l,c,m,n},\ (c,m,n)\in Q^{IN}_l,\ l\in[\![2,k]\!] \\
& z^R_{l,c,m,n},\ (c,m,n)\in Q^{IN}_l,\ l\in[\![2,k]\!] \\
& (z^M_{l,c,m,n})_a,\ (c,m,n,a)\in S^{IN}_l\times[\![0,k^M_l k^M_l]\!],\ l\in[\![2,k]\!] \\
& (\bar{z}_{l,c,m,n})_a,\ (c,m,n,a)\in S^{IN}_l\times[\![0,k^M_l k^M_l-1]\!],\ l\in[\![2,k]\!] \\
& (z'_{l,c,m,n})_a,\ (c,m,n,a)\in S^{IN}_l\times[\![0,k^M_l k^M_l-1]\!],\ l\in[\![2,k]\!] \\
& \tilde{\mathbf{z}}_l\in\mathbb{R}^{a_l},\ l\in[\![k,k+t-1]\!] \\
& \hat{\mathbf{z}}'_l\in\mathbb{R}^{a_l},\ l\in[\![k+1,k+t]\!] \qquad (22)
\end{aligned}
$$

- ReLU (In convolution part)

$$\left.\begin{cases} -z^R_{l,c,m,n}=0 \Rightarrow \mu^R_{l,c,m,n}\in\mathbb{R} & ,(c,m,n)\in\mathcal{I}^-_l \\ \hat{z}_{l,c,m,n}-z^R_{l,c,m,n}=0 \Rightarrow \tau^R_{l,c,m,n}\in\mathbb{R} & ,(c,m,n)\in\mathcal{I}^+_l \\ \left.\begin{array}{l} -z^R_{l,c,m,n}\leq 0 \Rightarrow \mu^R_{l,c,m,n}\in\mathbb{R}_{\geq 0} \\ \hat{z}_{l,c,m,n}-z^R_{l,c,m,n}\leq 0 \Rightarrow \tau^R_{l,c,m,n}\in\mathbb{R}_{\geq 0} \\ (\hat{u}_{l,c,m,n}-\hat{l}_{l,c,m,n})z^R_{l,c,m,n}-\hat{u}_{l,c,m,n}\hat{z}_{l,c,m,n}+\hat{u}_{l,c,m,n}\hat{l}_{l,c,m,n}\leq 0 \Rightarrow \lambda^R_{l,c,m,n}\in\mathbb{R}_{\geq 0} \end{array}\right\} & ,(c,m,n)\in\mathcal{I}_l \end{cases}\right\}, l\in[\![2,k]\!] \tag{24}$$

- Maxpool

$$\begin{aligned} &(z^M_{l,c,m,n})_0=0\Rightarrow(\nu^M_{l,c,m,n})_0\in\mathbb{R},\ (c,m,n)\in S^{IN}_l,\ l\in[\![2,k]\!] \\ &(\bar{z}_{l,c,m,n})_{ik^M_l+j}-[z^R_{l,c,s^M_l m+i,s^M_l n+j}-(z^M_{l,c,m,n})_{ik^M_l+j}]=0\Rightarrow(\kappa_{l,c,m,n})_{ik^M_l+j}\in\mathbb{R},\ \begin{cases} l\in[\![2,k]\!], \\ (c,m,n)\in S^{IN}_l, \\ i\in[\![0,k^M_l-1]\!], \\ j\in[\![0,k^M_l-1]\!] \end{cases} \\ &\left\{\begin{cases} -(z'_{l,c,m,n})_a=0\Rightarrow(\mu^M_{l,c,m,n})_a\in\mathbb{R} & ,(c,m,n,a)\in\bar{\mathcal{I}}^-_l \\ (\bar{z}_{l,c,m,n})_a-(z'_{l,c,m,n})_a=0\Rightarrow(\tau^M_{l,c,m,n})_a\in\mathbb{R} & ,(c,m,n,a)\in\bar{\mathcal{I}}^+_l \\ \left.\begin{array}{l} -(z'_{l,c,m,n})_a\leq 0\Rightarrow(\mu^M_{l,c,m,n})_a\in\mathbb{R}_{\geq 0} \\ (\bar{z}_{l,c,m,n})_a-(z'_{l,c,m,n})_a\leq 0\Rightarrow(\tau^M_{l,c,m,n})_a\in\mathbb{R}_{\geq 0} \\ [(\bar{u}_{l,c,m,n})_a-(\bar{l}_{l,c,m,n})_a](z'_{l,c,m,n})_a-(\bar{u}_{l,c,m,n})_a(\bar{z}_{l,c,m,n})_a+(\bar{u}_{l,c,m,n})_a(\bar{l}_{l,c,m,n})_a \\ \quad\leq 0\Rightarrow(\lambda^M_{l,c,m,n})_a\in\mathbb{R}_{\geq 0} \end{array}\right\} & ,(c,m,n,a)\in\bar{\mathcal{I}}_l \end{cases}\right\} \\ &\text{, where } l\in[\![2,k]\!] \\ &(z^M_{l,c,m,n})_{a+1}-[(z'_{l,c,m,n})_a+(z^M_{l,c,m,n})_a]=0\Rightarrow(\rho_{l,c,m,n})_{a+1}\in\mathbb{R},\ (c,m,n,a)\in S^{IN}_l\times[\![0,k^M_l k^M_l-1]\!],\ l\in[\![2,k]\!] \\ &z_{l,c,m,n}-(z^M_{l,c,m,n})_{k^M_l k^M_l}=0\Rightarrow\beta_{l,c,m,n}\in\mathbb{R},\ (c,m,n)\in S^{IN}_l,\ l\in[\![2,k]\!] \end{aligned} \tag{25}$$

- Flatten

$$\tilde{z}_{k,a}-\sum_{(c,m,n)\in S^{IN}_k}(\hat{W}_{c,m,n})_a z_{k,c,m,n}=0\Rightarrow\gamma_{k,a}\in\mathbb{R},\ a\in[\![0,a_k-1]\!]$$

- Fully connected

$$\hat{\mathbf{z}}'_{l+1}-(\mathbf{W}_l\tilde{\mathbf{z}}_l+\mathbf{b}_l)=0\Rightarrow\boldsymbol{\nu}'_{l+1}\in\mathbb{R}^{a_{l+1}},\ l\in[\![k,k+t-1]\!]$$

- ReLU (In linear part)

$$\left.\begin{cases} -\tilde{z}_{l,a}=0\Rightarrow\mu'_{l,a}\in\mathbb{R} & ,a\in\hat{\mathcal{I}}^-_l \\ \hat{z}'_{l,a}-\tilde{z}_{l,a}=0\Rightarrow\tau'_{l,a}\in\mathbb{R} & ,a\in\hat{\mathcal{I}}^+_l \\ \left.\begin{array}{l} -\tilde{z}_{l,a}\leq 0\Rightarrow\mu'_{l,a}\in\mathbb{R}_{\geq 0} \\ \hat{z}'_{l,a}-\tilde{z}_{l,a}\leq 0\Rightarrow\tau'_{l,a}\in\mathbb{R}_{\geq 0} \\ (u'_{l,a}-l'_{l,a})\tilde{z}_{l,a}-u'_{l,a}\hat{z}'_{l,a}+u'_{l,a}l'_{l,a}\leq 0\Rightarrow\lambda'_{l,a}\in\mathbb{R}_{\geq 0} \end{array}\right\} & ,a\in\hat{\mathcal{I}}_l \end{cases}\right\}, l\in[\![k+1,k+t-1]\!] \tag{26}$$

Let Θ denote the collection of dual variables, the dual problem of can be derived as (27) with dual variables (28).

maximize

$$g(\Theta)=\sum_{(c,m,n)\in S_1} -x_{c,m,n}(\xi^+_{c,m,n}-\xi^-_{c,m,n})-\sum_{(c,m,n)\in S_1}\epsilon_{c,m,n}(\xi^+_{c,m,n}+\xi^-_{c,m,n})$$
$$+\sum_{\substack{l\in[\![2,k]\!]\\(c,m,n)\in Q_l^{IN}}}\lambda^R_{l,c,m,n}(\hat{u}_{l,c,m,n}\hat{l}_{l,c,m,n})+\sum_{\substack{l\in[\![2,k]\!]\\(c,m,n)\in S_l^{IN}\\a\in[\![0,k_l^Mk_l^M-1]\!]}}(\lambda^M_{l,c,m,n})_a[(\bar{u}_{l,c,m,n})_a(\bar{l}_{l,c,m,n})_a]$$
$$+\sum_{l\in[\![k+1,k+t-1]\!]}{\boldsymbol{\lambda}'_l}^T[\boldsymbol{u}'_l\odot\boldsymbol{l}'_l]-\sum_{\substack{l\in[\![1,k-1]\!]\\(c,m,n)\in Q_{l+1}^{IN}}}\nu_{l+1,c,m,n}b_{l,c}-\sum_{l\in[\![k,k+t-1]\!]}{\boldsymbol{\nu}'_{l+1}}^T\boldsymbol{b}_l \tag{27a}$$

subject to

$$\boldsymbol{\nu}'_{k+t}=\boldsymbol{d} \tag{27b}$$
$$\xi^+_{c,m,n}-\xi^-_{c,m,n}+\nu^P_{1,c,m,n}=\sum_{(\hat{c},m',n')\in Q_2^{IN}}\nu_{2,\hat{c},m',n'}h_{1,\hat{c}}(c,m+p_1^{cv}-s_1^{cv}m',n+p_1^{cv}-s_1^{cv}n'),\ (c,m,n)\in S_1 \tag{27c}$$
$$\nu^P_{l,c,m,n}+\beta_{l,c,m,n}=\sum_{(\hat{c},m',n')\in Q_{l+1}^{IN}}\nu_{l+1,\hat{c},m',n'}h_{l,\hat{c}}(c,m+p_l^{cv}-s_l^{cv}m',n+p_l^{cv}-s_l^{cv}n'),\begin{cases}l\in[\![2,k-1]\!],\\(c,m,n)\in S_l\end{cases} \tag{27d}$$
$$\beta_{k,c,m,n}=\sum_{a=0}^{a_k-1}(\hat{W}_{c,m,n})_a\gamma_{k,a},\ (c,m,n)\in S_k^{IN} \tag{27e}$$
$$\nu_{l,c,m,n}=-\tau^R_{l,c,m,n}+\lambda^R_{l,c,m,n}\hat{u}_{l,c,m,n},\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in Q_l^{IN}\end{cases} \tag{27f}$$
$$-\mu^R_{l,c,m,n}-\tau^R_{l,c,m,n}+\lambda^R_{l,c,m,n}(\hat{u}_{l,c,m,n}-\hat{l}_{l,c,m,n})=\sum_{\substack{(i,j)\in[\![0,k_l^M-1]\!]\times[\![0,k_l^M-1]\!]\\(c,\frac{m-i}{s_l^M},\frac{n-j}{s_l^M})\in S_l^{IN}}}(\kappa_{l,c,\frac{m-i}{s_l^M},\frac{n-j}{s_l^M}})_{ik_l^M+j},\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in Q_l^{IN}\end{cases} \tag{27g}$$
$$(\nu^M_{l,c,m,n})_0=-(\kappa_{l,c,m,n})_0+(\rho_{l,c,m,n})_1,\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in S_l^{IN}\end{cases} \tag{27h}$$
$$(\rho_{l,c,m,n})_{k_l^Mk_l^M}=\beta_{l,c,m,n},\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in S_l^{IN}\end{cases} \tag{27i}$$
$$(\rho_{l,c,m,n})_a=(\rho_{l,c,m,n})_{a+1}-(\kappa_{l,c,m,n})_a,\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in S_l^{IN},\\a\in[\![1,k_l^Mk_l^M-1]\!]\end{cases} \tag{27j}$$
$$(\kappa_{l,c,m,n})_a=-(\tau^M_{l,c,m,n})_a+(\lambda^M_{l,c,m,n})_a(\bar{u}_{l,c,m,n})_a,\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in S_l^{IN},\\a\in[\![0,k_l^Mk_l^M-1]\!]\end{cases} \tag{27k}$$
$$(\rho_{l,c,m,n})_{a+1}=-[(\mu^M_{l,c,m,n})_a+(\tau^M_{l,c,m,n})_a]+(\lambda^M_{l,c,m,n})_a[(\bar{u}_{l,c,m,n})_a-(\bar{l}_{l,c,m,n})_a],\begin{cases}l\in[\![2,k]\!],\\(c,m,n)\in S_l^{IN},\\a\in[\![0,k_l^Mk_l^M-1]\!]\end{cases} \tag{27l}$$
$$\mathbf{W}_l^T\boldsymbol{\nu}'_{l+1}=-(\boldsymbol{\mu}'_l+\boldsymbol{\tau}'_l)+\boldsymbol{\lambda}'_l\odot(\boldsymbol{u}'_l-\boldsymbol{l}'_l),\ l\in[\![k+1,k+t-1]\!] \tag{27m}$$
$$\boldsymbol{\nu}'_l=-\boldsymbol{\tau}'_l+\boldsymbol{\lambda}'_l\odot\boldsymbol{u}'_l,\ l\in[\![k+1,k+t-1]\!] \tag{27n}$$
$$\boldsymbol{\gamma}_k=\mathbf{W}_k^T\boldsymbol{\nu}'_{k+1} \tag{27o}$$
$$\tag{27p}$$

$$\begin{cases}\xi^+_{c,m,n}, \xi^-_{c,m,n} \in \mathbb{R}_{\geq 0}, (c,m,n) \in S_1^{IN} \\ \xi^+_{c,m,n}, \xi^-_{c,m,n} = 0, (c,m,n) \in S_1^{UD} \cup S_1^{LR}\end{cases} \tag{28a}$$

$$\left.\begin{cases}\nu^P_{l,c,m,n} = 0, (c,m,n) \in S_l^{IN} \\ \nu^P_{l,c,m,n} \in \mathbb{R}, (c,m,n) \in S_l^{UD} \cup S_l^{LR}\end{cases}\right\}, l \in [\![1, k-1]\!] \tag{28b}$$

$$\nu_{l,c,m,n} \in \mathbb{R}, (c,m,n) \in Q_l^{IN}, l \in [\![2,k]\!] \tag{28c}$$

$$\begin{cases}\mu^R_{l,c,m,n} \in \mathbb{R}, \tau^R_{l,c,m,n} = 0, \lambda^R_{l,c,m,n} = 0 & ,(c,m,n) \in \mathcal{I}_l^- \\ \mu^R_{l,c,m,n} \geq 0, \tau^R_{l,c,m,n} \geq 0, \lambda^R_{l,c,m,n} \geq 0 & ,(c,m,n) \in \mathcal{I}_l \quad , l \in [\![2,k]\!] \\ \mu^R_{l,c,m,n} = 0, \tau^R_{l,c,m,n} \in \mathbb{R}, \lambda^R_{l,c,m,n} = 0 & ,(c,m,n) \in \mathcal{I}_l^+\end{cases} \tag{28d}$$

$$(\nu^M_{l,c,m,n})_0 \in \mathbb{R}, (c,m,n) \in S_l^{IN}, l \in [\![2,k]\!] \tag{28e}$$

$$(\kappa_{l,c,m,n})_a \in \mathbb{R}, (c,m,n,a) \in S_l^{IN} \times [\![0, k_l^M k_l^M - 1]\!], l \in [\![2,k]\!] \tag{28f}$$

$$\left.\begin{cases}(\mu^M_{l,c,m,n})_a \in \mathbb{R}, (\tau^M_{l,c,m,n})_a = 0, (\lambda^M_{l,c,m,n})_a = 0 & ,(c,m,n,a) \in \bar{\mathcal{I}}_l^- \\ (\mu^M_{l,c,m,n})_a \geq 0, (\tau^M_{l,c,m,n})_a \geq 0, (\lambda^M_{l,c,m,n})_a \geq 0 & ,(c,m,n,a) \in \bar{\mathcal{I}}_l \\ (\mu^M_{l,c,m,n})_a = 0, (\tau^M_{l,c,m,n})_a \in \mathbb{R}, (\lambda^M_{l,c,m,n})_a = 0 & ,(c,m,n,a) \in \bar{\mathcal{I}}_l^+\end{cases}\right\}, l \in [\![2,k]\!] \tag{28g}$$

$$(\rho_{l,c,m,n})_a \in \mathbb{R}, (c,m,n,a) \in S_l^{IN} \times [\![1, k_l^M k_l^M]\!], l \in [\![2,k]\!] \tag{28h}$$

$$\left.\begin{cases}\beta_{l,c,m,n} \in \mathbb{R}, (c,m,n) \in S_l^{IN} \\ \beta_{l,c,m,n} = 0, (c,m,n) \in S_l^{UD} \cup S_l^{LR}\end{cases}\right\}, l \in [\![2,k]\!] \tag{28i}$$

$$\boldsymbol{\gamma}_k \in \mathbb{R}^{a_k} \tag{28j}$$

$$\boldsymbol{\nu}'_l \in \mathbb{R}^{a_l}, l \in [\![k+1, k+t]\!] \tag{28k}$$

$$\left.\begin{cases}\mu'_{l,a} \in \mathbb{R}, \tau'_{l,a} = 0, \lambda'_{l,a} = 0 & ,a \in \hat{\mathcal{I}}_l^- \\ \mu'_{l,a} \geq 0, \tau'_{l,a} \geq 0, \lambda'_{l,a} \geq 0 & ,a \in \hat{\mathcal{I}}_l \\ \mu'_{l,a} = 0, \tau'_{l,a} \in \mathbb{R}, \lambda'_{l,a} = 0 & ,a \in \hat{\mathcal{I}}_l^+\end{cases}\right\}, l \in [\![k+1, k+t-1]\!] \tag{28l}$$

A.3.3 CNN DUAL NETWORK

In this section, we introduce how to successively solve the optimal dual variables in a way similar to a feed-forward CNN, which is referred as the dual network. We start from directly solving $\nu'_{k+t} = d$ (cf. (27b)). By substituting (28l) into (27m) and (27n), we can easily express $\mu'_{l,a}, \tau'_{l,a}, \lambda'_{l,a}, \ \nu'_{l,a}$ in terms of $W_l^T \nu'_{l+1}$ when a either belongs to $\hat{\mathcal{I}}_l^-$ or $\hat{\mathcal{I}}_l^+$, namely

$$\begin{aligned}&\mu'_{l,a} = -[W_l^T \nu'_{l+1}]_a, \tau'_{l,a} = 0, \lambda'_{l,a} = 0, \nu'_{l,a} = 0 && , a \in \hat{\mathcal{I}}_l^- \\ &\mu'_{l,a} = 0, \tau'_{l,a} = -[W_l^T \nu'_{l+1}]_a, \lambda'_{l,a} = 0, \nu'_{l,a} = [W_l^T \nu'_{l+1}]_a && , a \in \hat{\mathcal{I}}_l^+ .\end{aligned}$$

For $a \in \hat{\mathcal{I}}_l$, impose complementary slackness constraints to (26) yields

$$\mu'_{l,a}(-\tilde{z}_{l,a}) = 0 \tag{29a}$$

$$\tau'_{l,a}(\hat{z}'_{l,a} - \tilde{z}_{l,a}) = 0 \tag{29b}$$

$$\lambda'_{l,a}[(u'_{l,a} - l'_{l,a})\tilde{z}_{l,a} - u'_{l,a}\hat{z}'_{l,a} + u'_{l,a} l'_{l,a}] = 0 \tag{29c}$$

Note that as illustrated in Fig. 10, either $\lambda'_{l,a} = 0$ or $\mu'_{l,a} = \tau'_{l,a} = 0$. Recall that $\mu'_{l,a}, \tau'_{l,a}, \lambda'_{l,a}$ are non-negative (cf. (28l)). If $\lambda'_{l,a} = 0$, then by (27m),

$$\begin{aligned}&\mu'_{l,a} + \tau'_{l,a} = -(W_l^T \nu'_{l+1})_a = [(W_l^T \nu'_{l+1})_a]_- \\ &\lambda'_{l,a}(u'_{l,a} - l'_{l,a}) = 0 = [(W_l^T \nu'_{l+1})_a]_+\end{aligned}$$

where we use notation $[\xi]_+ = \xi \vee 0$ and $[\xi]_- = (-\xi) \vee 0$. If $\mu'_{l,a} = \tau'_{l,a} = 0$, then by (27m)

$$\begin{aligned}&\lambda'_{l,a}(u'_{l,a} - l'_{l,a}) = (W_l^T \nu'_{l+1})_a = [(W_l^T \nu'_{l+1})_a]_+ \\ &\mu'_{l,a} + \tau'_{l,a} = 0 = [(W_l^T \nu'_{l+1})_a]_-\end{aligned} .$$

As such, for $a\in\hat{\mathcal{I}}_l$, we can write

$$\begin{aligned}(\mu'_l+\tau'_l)_a&=[(W_l^T\nu'_{l+1})_a]_-\\(\lambda'_l\odot(u'_l-l'_l))_a&=[(W_l^T\nu'_{l+1})_a]_+\end{aligned}. \tag{30}$$

Hence, by introducing additional variables $\alpha'_{l,a}\in[0,1]$, we can rewrite (30) and (27n) as

$$\begin{aligned}\tau'_{l,a}&=\alpha'_{l,a}[(W_l^T\nu'_{l+1})_a]_-\\\mu'_{l,a}&=(1-\alpha'_{l,a})[(W_l^T\nu'_{l+1})_a]_-\\\lambda'_{l,a}&=\frac{1}{u'_{l,a}-l'_{l,a}}[(W_l^T\nu'_{l+1})_a]_+\\\nu'_{l,a}&=\frac{u'_{l,a}}{u'_{l,a}-l'_{l,a}}[(W_l^T\nu'_{l+1})_a]_+-\alpha'_{l,a}[(W_l^T\nu'_{l+1})_a]_-\end{aligned}. \tag{31}$$

Thus, with ν'_{l+1} at hand and introduce variable $\hat{\nu}'_{l,a}=(W_l^T\nu'_{l+1})_a$, we can solve ν'_l as follows:

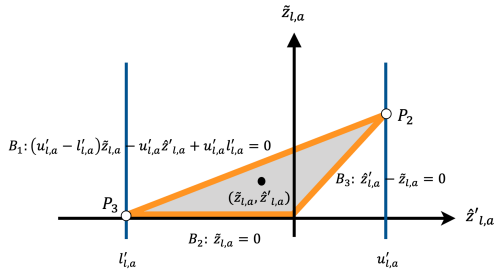

Figure 10: Recall (17c) that the primal feasible solution for $(\hat{z}'_{l,a},\tilde{z}_{l,a})$ must lie within the grayed area (not including points P_2 and P_3). If $(\hat{z}'_{l,a},\tilde{z}_{l,a})$ does not lie on boundary B_1, then (29c) implies $\lambda'_{l,a}=0$. If $(\hat{z}'_{l,a},\tilde{z}_{l,a})$ lies on B_1, then (29a) and (29b) imply $\mu'_{l,a}=\tau'_{l,a}=0$.

$$\begin{aligned}\mu'_{l,a}&=\begin{cases}-\hat{\nu}'_{l,a} & ,a\in\hat{\mathcal{I}}_l^-\\0 & ,a\in\hat{\mathcal{I}}_l^+\\(1-\alpha'_{l,a})[\hat{\nu}'_{l,a}]_- & ,a\in\hat{\mathcal{I}}_l\end{cases}\\\tau'_{l,a}&=\begin{cases}0 & ,a\in\hat{\mathcal{I}}_l^-\\-\hat{\nu}'_{l,a} & ,a\in\hat{\mathcal{I}}_l^+\\\alpha'_{l,a}[\hat{\nu}'_{l,a}]_- & ,a\in\hat{\mathcal{I}}_l\end{cases}\\\lambda'_{l,a}&=\begin{cases}0 & ,a\in\hat{\mathcal{I}}_l^-\\0 & ,a\in\hat{\mathcal{I}}_l^+\\\frac{1}{u'_{l,a}-l'_{l,a}}[\hat{\nu}'_{l,a}]_+ & ,a\in\hat{\mathcal{I}}_l\end{cases}\end{aligned} \tag{32}$$

$$\nu'_{l,a}=\left\{\begin{array}{ll}0 & ,a\in\hat{\mathcal{I}}_l^-\\\hat{\nu}'_{l,a} & ,a\in\hat{\mathcal{I}}_l^+\\\frac{u'_{l,a}}{u'_{l,a}-l'_{l,a}}[\hat{\nu}'_{l,a}]_+-\alpha'_{l,a}[\hat{\nu}'_{l,a}]_- & ,a\in\hat{\mathcal{I}}_l\end{array}\right\} \tag{33}$$

$, l\in[\![k+1,k+t-1]\!]$

After obtaining $\boldsymbol{\nu}'_{k+1}$, we can use it to calculate $\boldsymbol{\gamma}_k$ by (27o) and then use (27e) to reshape $\boldsymbol{\gamma}_k$ as a three-dimensional map $\boldsymbol{\beta}_k$. The constraints (27i) also indicates that the value of $(\rho_{l,c,m,n})_{k_l^M k_l^M}$ is equal to $\beta_{l,c,m,n}$. To sum up, with currently available information, we can compute

$$\gamma_k=W_k^T\boldsymbol{\nu}'_{k+1} \tag{34}$$

$$\beta_{k,c,m,n}=\sum_{a=0}^{a_k-1}(\hat{W}_{c,m,n})_a\gamma_{k,a},\ (c,m,n)\in S_k^{IN} \tag{35}$$

$$(\rho_{k,c,m,n})_{k_l^M k_l^M}=\beta_{k,c,m,n},\ (c,m,n)\in S_l^{IN}. \tag{36}$$

We then use $(\rho_{k,c,m,n})_{k_k^M k_k^M}$ to solve other dual variables. Observe that (27k), (27l) and (28c) take similar form as (27m), (27n) and (28l). Thus, by similar derivation, we may introduce additional variables $(\alpha^M_{l,c,m,n})_a\in[0,1]$ and obtain (37), (38).

$$\begin{aligned}
(\mu^M_{l,c,m,n})_a&=\begin{cases}-(\rho_{l,c,m,n})_{a+1} & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^-\\ 0 & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^+\\ (1-(\alpha^M_{l,c,m,n})_a)[(\rho_{l,c,m,n})_{a+1}]_- & ,(c,m,n,a)\in\bar{\mathcal{I}}_l\end{cases}\\
(\tau^M_{l,c,m,n})_a&=\begin{cases}0 & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^-\\ -(\rho_{l,c,m,n})_{a+1} & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^+\\ (\alpha^M_{l,c,m,n})_a[(\rho_{l,c,m,n})_{a+1}]_- & ,(c,m,n,a)\in\bar{\mathcal{I}}_l\end{cases}\\
(\lambda^M_{l,c,m,n})_a&=\begin{cases}0 & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^-\\ 0 & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^+\\ \dfrac{1}{(\bar{u}_{l,c,m,n})_a-(\bar{l}_{l,c,m,n})_a}[(\rho_{l,c,m,n})_{a+1}]_+ & ,(c,m,n,a)\in\bar{\mathcal{I}}_l\end{cases}
\end{aligned} \tag{37}$$

$$(\kappa_{l,c,m,n})_a=\left\{\begin{array}{ll}0 & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^-\\ (\rho_{l,c,m,n})_{a+1} & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^+\\ \dfrac{(\bar{u}_{l,c,m,n})_a}{(\bar{u}_{l,c,m,n})_a-(\bar{l}_{l,c,m,n})_a}[(\rho_{l,c,m,n})_{a+1}]_+-(\alpha^M_{l,c,m,n})_a[(\rho_{l,c,m,n})_{a+1}]_- & ,(c,m,n,a)\in\bar{\mathcal{I}}_l\end{array}\right\} \tag{38}$$

Thus, $(\rho_{l,c,m,n})_a$ can be obtained by (27j), namely:

$$(\rho_{l,c,m,n})_a=(\rho_{l,c,m,n})_{a+1}-(\kappa_{l,c,m,n})_a\ . \tag{39}$$

With $(\kappa_{l,c,m,n})_a$ at hand, we may solve other dual variables as elaborated below. We first define

$$\begin{aligned}
\hat{\kappa}_{l,c,m,n}=&\sum_{\substack{(i,j)\in[\![0,k_l^M-1]\!]\times[\![0,k_l^M-1]\!]\\ (c,\frac{m-i}{s_l^M},\frac{n-j}{s_l^M})\in S_l^{IN}}}(\kappa_{l,c,\frac{m-i}{s_l^M},\frac{n-j}{s_l^M}})_{ik_l^M+j},\\
&l\in[\![2,k]\!],\ (c,m,n)\in Q_l^{IN}\ .
\end{aligned} \tag{40}$$

Observe that (27f), (27g) and (28d) take similar form as (27m), (27n) and (28l). Thus by similar derivation, we may introduce additional variables $\alpha^R_{l,c,m,n}\in[0,1]$ and obtain (41) and (42)

$$\begin{aligned}
\mu^R_{l,c,m,n}&=\begin{cases}-\hat{\kappa}_{l,c,m,n} & ,(c,m,n)\in\mathcal{I}_l^-\\ 0 & ,(c,m,n)\in\mathcal{I}_l^+\\ (1-\alpha^R_{l,c,m,n})[\hat{\kappa}_{l,c,m,n}]_- & ,(c,m,n)\in\mathcal{I}_l\end{cases}\\
\tau^R_{l,c,m,n}&=\begin{cases}0 & ,(c,m,n)\in\mathcal{I}_l^-\\ -\hat{\kappa}_{l,c,m,n} & ,(c,m,n)\in\mathcal{I}_l^+\\ \alpha^R_{l,c,m,n}[\hat{\kappa}_{l,c,m,n}]_- & ,(c,m,n)\in\mathcal{I}_l\end{cases}\\
\lambda^R_{l,c,m,n}&=\begin{cases}0 & ,(c,m,n)\in\mathcal{I}_l^-\\ 0 & ,(c,m,n)\in\mathcal{I}_l^+\\ \dfrac{1}{\hat{u}_{l,c,m,n}-\hat{l}_{l,c,m,n}}[\hat{\kappa}_{l,c,m,n}]_+ & ,(c,m,n)\in\mathcal{I}_l\end{cases}
\end{aligned} \tag{41}$$

$$\nu_{l,c,m,n}=\left\{\begin{array}{ll}0 & ,(c,m,n)\in\mathcal{I}_l^-\\ \hat{\kappa}_{l,c,m,n} & ,(c,m,n)\in\mathcal{I}_l^+\\ \dfrac{\hat{u}_{l,c,m,n}}{\hat{u}_{l,c,m,n}-\hat{l}_{l,c,m,n}}[\hat{\kappa}_{l,c,m,n}]_+-\alpha^R_{l,c,m,n}[\hat{\kappa}_{l,c,m,n}]_- & ,(c,m,n)\in\mathcal{I}_l\end{array}\right\} \tag{42}$$

With $\nu_{l+1,c,m,n}$ at hand, we may solve $\beta_{l,c,m,n}$ with (27d) (Recall that $\nu^P_{l,c,m,n}=0$ when $(c,m,n)\in S^{IN}_l$ by (27g))

$$\beta_{l,c,m,n}=\hat{\nu}_{l,c,m,n},\, l\in[\![2,k-1]\!],(c,m,n)\in S^{IN}_l \tag{43}$$

$$\hat{\nu}_{l,c,m,n}=\sum_{(\hat{c},m',n')\in Q^{IN}_{l+1}} \nu_{l+1,\hat{c},m',n'} h_{l,\hat{c}}(c,m+p^{cv}_l-s^{cv}_l m',n+p^{cv}_l-s^{cv}_l n'),\, l\in[\![1,k-1,\, (c,m,n)\in S^{IN}_l \tag{44}$$

where we define $\hat{\nu}_{l,c,m,n}$ in (44). We thus obtain $(\rho_{l-1,c,m,n})_{k^M_l k^M_l}$ by (27i). To sum up, we may consecutively solve the dual variables associated with the convolution layers in a layer-by-layer manner:

$$\begin{aligned}&\beta_{l,c,m,n}\xrightarrow{(36)}(\rho_{l,c,m,n})_{k^M_l k^M_l}\xrightarrow{(38),(39)}(\kappa_{l,c,m,n})_a,\,(\rho_{l,c,m,n})_a\\ &\xrightarrow{(40)}\hat{\kappa}_{l,c,m,n}\xrightarrow{(42)}\nu_{l,c,m,n}\xrightarrow{(44)}\hat{\nu}_{l-1,c,m,n}\xrightarrow{(43)}\beta_{l-1,c,m,n}\,.\end{aligned} \tag{45}$$

After obtaining ν_2, recall (27c) as well as constraints (28a) and (28b), it follows that

$$\xi^+_{c,m,n}-\xi^-_{c,m,n}=\hat{\nu}_{1,c,m,n},\,(c,m,n)\in S^{IN}_1, \tag{46}$$

where $\hat{\nu}_1$ is given by (44). Note that, by (28a)

$$\begin{aligned}&\xi^+_{c,m,n}\geq[\hat{\nu}_{1,c,m,n}]_+\\ &\xi^-_{c,m,n}\geq[\hat{\nu}_{1,c,m,n}]_-\end{aligned}\,. \tag{47}$$

As such, to maximize $-(\xi^+_{c,m,n}+\xi^-_{c,m,n})$ that appears in the objective function g (cf. (27a)), we choose $\xi^+_{c,m,n}=[\hat{\nu}_{1,c,m,n}]_+$ and $\xi^-_{c,m,n}=[\hat{\nu}_{1,c,m,n}]_-$. Finally, we can simplify (27) as (48), which we refer as the CNN dual network.

maximize

$$g(\Theta)=\sum_{(c,m,n)\in S_1^{IN}} -x_{c,m,n}\hat{\nu}_{1,c,m,n}-\sum_{(c,m,n)\in S_1^{IN}}\epsilon_{c,m,n}|\hat{\nu}_{1,c,m,n}|$$
$$-\sum_{l=1}^{k-1}\sum_{(c,m,n)\in F_{l+1}^{IN}}\nu_{l+1,c,m,n}b_{l,c}+\sum_{l=2}^{k}\sum_{(c,m,n)\in\mathcal{I}_l}\hat{l}_{l,c,m,n}[\nu_{l,c,m,n}]_+ \tag{48a}$$
$$+\sum_{l=2}^{k}\sum_{(c,m,n,a)\in\bar{\mathcal{I}}_l}(\bar{l}_{l,c,m,n})_a[(\kappa_{l,c,m,n})_a]_+-\sum_{l=k}^{k+t-1}{\nu'_{l+1}}^T b_l+\sum_{l=k+1}^{k+t-1}\sum_{a\in\hat{\mathcal{I}}_l}l'_{l,a}[\nu'_{l,a}]_+$$

subject to

$$\boldsymbol{\nu}'_{k+t}=\boldsymbol{d} \tag{48b}$$

$$\hat{\nu}'_{l,a}=(\boldsymbol{W}_l^T\boldsymbol{\nu}'_{l+1})_a,\ \begin{cases}l\in[\![k+1,k+t-1]\!],\\ a\in[\![0,a_l-1]\!]\end{cases} \tag{48c}$$

$$\nu'_{l,a}=\begin{cases}0 & ,a\in\hat{\mathcal{I}}_l^-\\ \hat{\nu}'_{l,a} & ,a\in\hat{\mathcal{I}}_l^+\\ \dfrac{u'_{l,a}}{u'_{l,a}-l'_{l,a}}[\hat{\nu}'_{l,a}]_+-\alpha'_{l,a}[\hat{\nu}'_{l,a}]_- & ,a\in\hat{\mathcal{I}}_l\end{cases},l\in[\![k+1,k+t-1]\!] \tag{48d}$$

$$\boldsymbol{\gamma}_k=\boldsymbol{W}_k^T\boldsymbol{\nu}'_{k+1} \tag{48e}$$

$$\begin{cases}\beta_{k,c,m,n}=\sum_{a=0}^{a_k-1}(\hat{W}_{c,m,n})_a\gamma_{k,a} & ,(c,m,n)\in S_k^{IN}\\ \beta_{l,c,m,n}=\hat{\nu}_{l,c,m,n} & ,\begin{cases}l\in[\![2,k-1]\!],\\ (c,m,n)\in S_l^{IN}\end{cases}\end{cases} \tag{48f}$$

$$(\rho_{l,c,m,n})_{k_l^M k_l^M}=\beta_{l,c,m,n},\ \begin{cases}l\in[\![2,k]\!],\\ (c,m,n)\in S_l^{IN}\end{cases} \tag{48g}$$

$$(\kappa_{l,c,m,n})_a=\begin{cases}0 & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^-\\ (\rho_{l,c,m,n})_{a+1} & ,(c,m,n,a)\in\bar{\mathcal{I}}_l^+\\ \dfrac{(\bar{u}_{l,c,m,n})_a}{(\bar{u}_{l,c,m,n})_a-(\bar{l}_{l,c,m,n})_a}[(\rho_{l,c,m,n})_{a+1}]_+-(\alpha^M_{l,c,m,n})_a[(\rho_{l,c,m,n})_{a+1}]_- & ,(c,m,n,a)\in\bar{\mathcal{I}}_l\end{cases}$$
$$,l\in[\![2,k]\!] \tag{48h}$$

$$(\rho_{l,c,m,n})_a=(\rho_{l,c,m,n})_{a+1}-(\kappa_{l,c,m,n})_a,\ \begin{cases}l\in[\![2,k]\!],\\ (c,m,n)\in S_l^{IN},\\ a\in[\![1,k_l^Mk_l^M-1]\!]\end{cases} \tag{48i}$$

$$\hat{\kappa}_{l,c,m,n}=\sum_{\substack{(i,j)\in[\![0,k_l^M-1]\!]\times[\![0,k_l^M-1]\!]\\ (c,\frac{m-i}{s_l^M},\frac{n-j}{s_l^M})\in S_l^{IN}}}(\kappa_{l,c,\frac{m-i}{s_l^M},\frac{n-j}{s_l^M}})_{ik_l^M+j},\ \begin{cases}l\in[\![2,k]\!],\\ (c,m,n)\in Q_l^{IN}\end{cases} \tag{48j}$$

$$\nu_{l,c,m,n}=\begin{cases}0 & ,(c,m,n)\in\mathcal{I}_l^-\\ \hat{\kappa}_{l,c,m,n} & ,(c,m,n)\in\mathcal{I}_l^+\\ \dfrac{\hat{u}_{l,c,m,n}}{\hat{u}_{l,c,m,n}-\hat{l}_{l,c,m,n}}[\hat{\kappa}_{l,c,m,n}]_+-\alpha^R_{l,c,m,n}[\hat{\kappa}_{l,c,m,n}]_- & ,(c,m,n)\in\mathcal{I}_l\end{cases},\ l\in[\![2,k]\!] \tag{48k}$$

$$\hat{\nu}_{l,c,m,n}=\sum_{(\hat{c},m',n')\in F_{l+1}^{IN}}\nu_{l+1,\hat{c},m',n'}h_{l,\hat{c}}(c,m+p_l^{cv}-s_l^{cv}m',n+p_l^{cv}-s_l^{cv}n'),\ \begin{cases}l\in[\![1,k-1]\!]\\ (c,m,n)\in S_l^{IN}\end{cases} \tag{48l}$$

A.4 BOUND ANALYSIS FOR INTERMEDIATE LAYERS

From Fig. 3, we observe that the accuracy lower bound predicted by each verification method becomes looser as ϵ increases, and such phenomenon is also present in CAPM. As bounds estimated for the previous operations will affect the later operation's bound prediction, it is crucial for us to understand how the bounds estimated in each intermediate operation (e.g., convolution, ReLU, Maxpool) differ from the actual adversarial polytopes. As such we may identify the dominant factors that loosens the accuracy lower bound as a guidance for future improvements.

A.4.1 EVALUATE BOUND TIGHTNESS BY BOUND GAP

If the real upper/lower bounds for each neuron from an arbitrary adversarial example were known, then a comparison with those real bounds would demonstrate the accuracy of the predicted bounds for each intermediate operation. The gap between the upper and lower bounds for the real and predicted bounds are compared, which is described as

$$g_r{=}u_r{-}l_r \quad \text{and} \quad g_p{=}u_p{-}l_p$$

where u_r and l_r denote the real upper and lower bounds, and u_p and l_p denote the predicted upper and lower bounds. The difference in the bound gap $g_p{-}g_r$ is always non-negative and a perfectly predicted bound gives a zero bound gap difference. By reporting the real/predicted bound gaps averaged over all neurons pertaining to that operation, we can demonstrate how tight the predicted bounds are in general compared to the real bounds in each specific operation.

In reality, the real upper and lower bounds are not known. A Monte-Carlo simulation procedure for an adversary randomly draws a large amount of N_{adv} adversary examples from the adversarial polytope at the input layer following a uniform distribution. The N_{adv} randomly drawn adversarial examples are then fed into the network and the intermediate values for each neuron are computed. For each neuron, the maximum and minimum of the N_{adv} values as realized by the N_{adv} adversary examples are then computed as an estimate of the real upper and lower bounds.

Fig. 11 illustrates how the real upper and lower bounds are estimated through the simulated adversarial polytope. The gray square at the upper left corner is the adversarial polytope $\{x{+}\Delta{:}\|\Delta\|_\infty{\leq}\epsilon\}$ in some high dimensional input space, where the black dot represents the original image x. We draw N_{adv} points from the adversarial polytope, as indicated by the colorful symbols in the gray square, which are then fed into the network and resulted in N_{adv} feature maps after the convolution operation. For a specific neuron that corresponds to the (m,n)'th pixel in the c'th channel of the feature map, this amounts to N_{adv} values, to which the maximum $u^{(est)}_{c,m,n}$ and minimum $l^{(est)}_{c,m,n}$ are computed as an approximation to the real upper and lower bounds. We can then estimate the real bound gap as $g^{(est)}_{c,m,n}{=}u^{(est)}_{c,m,n}{-}l^{(est)}_{c,m,n}$ which is then compared with the predicted bound gap. However, since the Monte-Carlo Simulation do not reach a tight approximation, we leave the further discussion in appendix A.4.2.

A.4.2 LIMITATION ON APPROXIMATING ADVERSARIAL POLYTOPE WITH MONTE CARLO SIMULATION

In the previous section we described an intuitive and easy-to-implement Monte-Carlo simulation method to estimate the real upper and lower bounds of the intermediate adversarial polytopes at each operation. Several questions naturally arise: Under what circumstances does such Monte-Carlo simulation method yield a good estimate to the real upper and lower bounds to the adversarial polytopes? Is it possible that such Monte-Carlo simulation method can even replace the optimization-theoretic robustness verification methods proposed in literature as well as in this work? To answer such questions, we consider a simple neural network consisting of only one fully-connected layer, to which the lower bound of the j-th output node subject to adversarial examples with l_∞ norm-bounded constraint is described as the following optimization problem: (The upper bound can be formulated

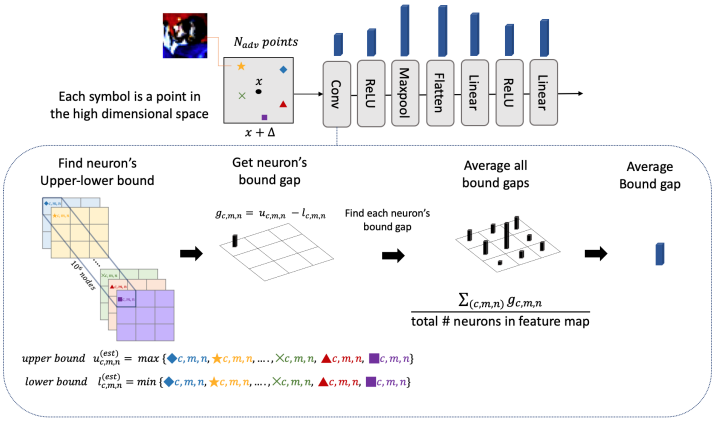

Figure 11: Estimating the real upper and lower bounds using a Monte Carlo simulation for an adversarial polytope.

in an analogous manner, namely by replacing e_j with $-e_j$ followed by a negation.)

$$\begin{aligned} \min_{\mathbf{y}} \quad & e_j^T y \\ \text{s.t.} \quad & z_1 \leq x+\epsilon \\ & z_1 \geq x-\epsilon \\ & y = W_1^T z_1 + b_1 \end{aligned} \tag{49}$$

Since (49) is a convex optimization problem, its primal and dual optimums must coincide. We use the CVX tool (Cvxpy) to find the primal optimum, and apply the dual network (cf. (Wong & Kolter, 2018), also in Sec. 2.2) to find dual optimum. The primal/dual optima are then compared with the bounds that re-estimated using the Monte-Carlo simulation method in Sec. A.4.1.

Fig. 12 shows a network consisting of one fully-connected linear layer with 784-dimensional inputs and 4-dimension outputs. N_{adv}=10^6 points are randomly sampled from the 784-dimensional cube $[0,1)^{784}$, and set ϵ=0.1, which corresponds to a large adversarial polytope, using the experimental settings in DeepPoly (Singh et al., 2019a). Fig. 12 shows that both the primal and dual optima coincide, so there is a perfect zero duality gap, as expected for a convex optimization problem. However, there is a significant gap between the primal/dual optima and the bound that is estimated using a Monte-Carlo simulation. This demonstrates that even 10^6 Monte-Carlo simulations are not sufficient to properly represent the complicated details of an actual adversarial polytope that may affect the bounds. This is intuitively reasonable because sampling the corners of the input adversarial polytope (which is a 784-dimensional cube) requires $2^{784} \approx 10^{236}$ samples which is not practically possible. If the input dimension is small (3 dimensions), then the bound that is estimated using a Monte-Carlo simulation coincides with the primal and dual optima, as shown in Fig. 13. A Monte-Carlo simulation method may provide us some insight on how well the bound predicted by a verification method is (cf. Sec. A.4.1), but Monte-Carlo simulation does not accurately represent the complicated details of the adversarial polytope, especially in highly dimensional input settings, so the bound gap for the actual adversarial polytope is underestimated (cf. Sec. A.4.1). This further highlights the necessity of a provable robustness verification scheme as discussed in this work.

A.5 Network structure

For the CNNs verified in Sec. A.1 (namely convSmallMNIST, convMedMNIST, convBigMNIST, convSmallCIFAR10, convMedCIFAR10, and convBigCIFAR10), we add maxpool layers to the convSmall, convMed, and convBig counterparts in (Mirman et al., 2018) and slightly adjust the parameters of striding and padding to achieve similar number of parameters as in (Mirman et al., 2018). The detailed network architectures are listed below: **convSmallMNIST**
$input \rightarrow Conv_{2,1,2,1}$ 16×4×4 $\rightarrow Conv_{2,1,2,1}$ 32×4×4 $\rightarrow Flat \rightarrow FC$(800,100) $\rightarrow ReLU \rightarrow FC$(100,10) $\rightarrow output$

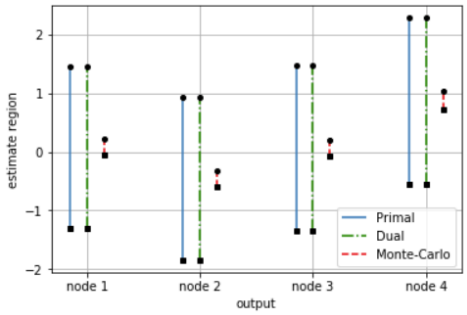

Figure 12: Comparison of (estimated) upper/lower bounds for the output nodes using the primal approach, the dual approach and a Monte-Carlo simulation with a one-layer network with 784-dimensional inputs. The endpoints of each line represent the upper (circle) and lower (square) bounds that are respectively predicted by each method.

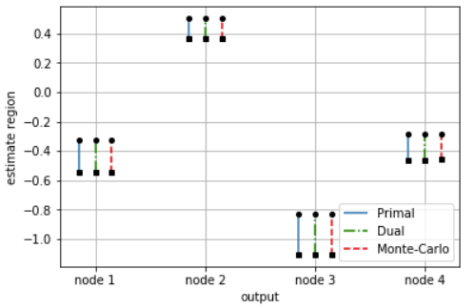

Figure 13: Comparison of (estimated) upper/lower bounds for the output nodes using the primal approach, the dual approach and a Monte-Carlo simulation with a one-layer network with 3-dimensional inputs. The endpoints of each line represent the upper (circle) and lower (square) bounds that are respectively predicted using each method.

convMedMNIST
$input \rightarrow Conv_{1,1,2,0}\ 16\times2\times2 \rightarrow Conv_{1,1,2,0}\ 32\times2\times2 \rightarrow Flat \rightarrow FC(1568,100) \rightarrow ReLU \rightarrow FC(100,10) \rightarrow output$

convBigMNIST
$input \rightarrow Conv_{1,1,2,1}\ 32\times3\times3 \rightarrow Conv_{1,0,2,1}\ 32\times4\times4 \rightarrow Conv_{1,0,2,1}\ 64\times3\times3 \rightarrow Conv_{2,0,2,0}\ 64\times4\times4 \rightarrow Flat \rightarrow FC(1024,512) \rightarrow ReLU \rightarrow FC(512,512) \rightarrow ReLU \rightarrow FC(512,10) \rightarrow output$

conv$_{\mathbf{S}}$ MNIST
$input \rightarrow Conv_{2,1,2,2}\ 16\times4\times4 \rightarrow Conv_{2,1,2,2}\ 32\times4\times4 \rightarrow Flat \rightarrow FC(32,24) \rightarrow ReLU \rightarrow FC(24,10) \rightarrow output$

conv$_{\mathbf{M}}$ MNIST
$input \rightarrow Conv_{2,1,2,2}\ 16\times4\times4 \rightarrow Conv_{2,1,2,2}\ 32\times4\times4 \rightarrow Conv_{1,1,2,2}\ 64\times2\times2 \rightarrow Flat \rightarrow FC(64,32) \rightarrow ReLU \rightarrow FC(32,10) \rightarrow output$

conv$_{\mathbf{L}}$ MNIST
$input \rightarrow Conv_{1,1,2,2}\ 32\times2\times2 \rightarrow Conv_{1,1,2,2}\ 64\times2\times2 \rightarrow Conv_{1,1,2,2}\ 128\times2\times2 \rightarrow Flat \rightarrow FC(2048,256) \rightarrow ReLU \rightarrow FC(256,10) \rightarrow output$

convSmallCIFAR10
$input \rightarrow Conv_{2,1,2,1}\ 16\times4\times4 \rightarrow Conv_{2,1,2,1}\ 32\times4\times4 \rightarrow Flat \rightarrow FC(1152,100) \rightarrow ReLU \rightarrow$

$FC(100,10) \rightarrow output$

convMedCIFAR10
$input \rightarrow Conv_{1,1,2,0}\ 16\times2\times2 \rightarrow Conv_{1,1,2,0}\ 32\times2\times2 \rightarrow Flat \rightarrow FC(2048,100) \rightarrow ReLU \rightarrow FC(100,10) \rightarrow output$

convBigCIFAR10
$input \rightarrow Conv_{1,1,2,1}\ 32\times3\times3 \rightarrow Conv_{1,1,2,0}\ 32\times4\times4 \rightarrow Conv_{1,0,2,1}\ 64\times3\times3 \rightarrow Conv_{2,0,2,1}\ 64\times4\times4 \rightarrow Flat \rightarrow FC(4096,512) \rightarrow ReLU \rightarrow FC(512,512) \rightarrow ReLU \rightarrow FC(512,10) \rightarrow output$
Here $Conv_{s,p,k,t}\ C\times W\times H$ represents a convolution layer consisting of a convolution operation followed by $ReLU$ and maxpool with kernel size $k\times k$ and stride t, where the convolution operation consists of C convolution kernels each of width W and height H along with stride s and padding p; $FC(M,N)$ represents a fully-connected layer with M input neurons and N output neurons We use open source repo provided in (Wong et al., 2020) for adversarial training and slightly modify its default parameters to fit our models. Specifically, the hyperparameters we used are illustrated in Table 6.

A.6 Reproducing the state-of-the-art method

As there were no reported empirical verification performance on maxpool-based CNNs, it is necessary to correctly reproduce and test the state-of-the-art methods on maxpool-based CNNs such as the models described in Sec. 1.1. We reproduce DeepPoly and DeepZ on the neural networks (which includes convolution and ReLU but without maxpool) following the settings in (Singh et al., 2019a; 2018) with the implementation provided in (Eth-Sri). Fig. A.6 indicates that the reproduced result is highly consistent with the reported results in (Singh et al., 2019a).

PRIMA is reproduced using a pretrained model from (Eth-Sri). Table 7 shows the reproduced results for verified robustness for the first 1000 samples and the average verification time for convSmall and convBig. The difference in runtime is attributed to the use of different hardware but the verified robustness is highly consistent with the reported results in (Müller et al., 2022).

Table 6: Hyperparameter for network training

Dataset	Method	epochs	batch size	learning rate	Optimizer	ϵ	alpha
MNIST	Normal	200	500	0.0001	Adam	-	-
	Fast	200	500	max 0.005	SGD	8/255	2/255
	PGD	200	500	max 0.005	SGD	8/255	2/255
CIFAR10	Normal	200	500	0.0001	Adam	-	-
	Fast	200	500	max 0.005	SGD	8/255	2/255
	PGD	200	500	max 0.005	SGD	2/255	2/255

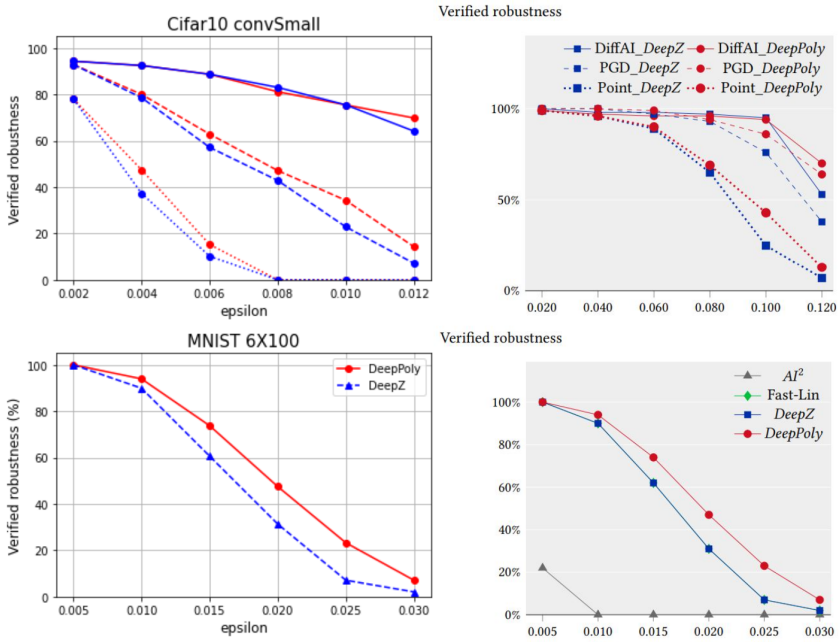

Figure 14: Verification results for DeepPoly and DeepZ using the settings in (Singh et al., 2019a; 2018) for the FFNNSmall and ConvSmall models (Singh et al., 2019a). The upper left figure shows the reproduced result for ConvSmall for Cifar10, which is consistent to the reported result (upper right) in the Fig.11(a) of (Singh et al., 2019a). The bottom left figure shows the reproduced result for FFNNSmall for MNIST $6X100$, which is consistent to the reported result (bottom right) in the Fig.5(a) of (Singh et al., 2019a).

Table 7: Reproduced result for PRIMA for convSmall and convBig.

Dataset	Model	Training	Accuracy	ϵ	Ver	Time	Ver (Reported)	Time (Reported)
MNIST	convSmall	NOR	980	0.120	650	132	640	51
	convBig	DiffAI	929	0.300	775	35.5	775	10.9
CIFAR10	convSmall	PGD	630	2/255	459	24.4	458	16