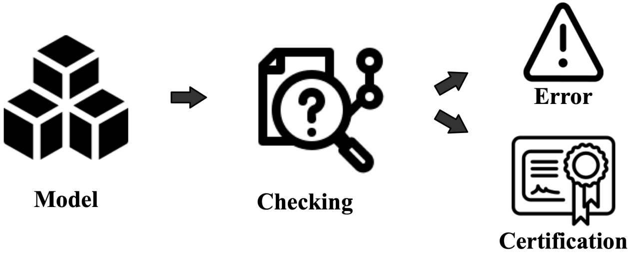
Robustness Verification for Deep Neural Networks

Guanqin Zhang [1,2] AP.Yulei Sui[1], Dr.Dilum Bandara[1,2], Dr.Shiping Chen[1,2]

[1]UNSW School of Computer Science and Engineering
[2]CSIRO DATA61

January 2, 2025

Background



Robustness verification aims to certify "specification guarantees" on model behaviors or find the errors of the model.

Robustness Formulation

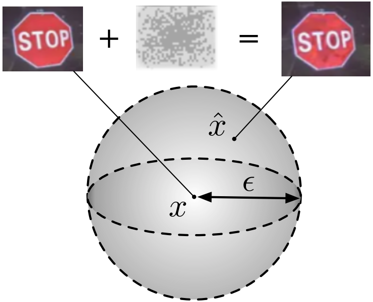


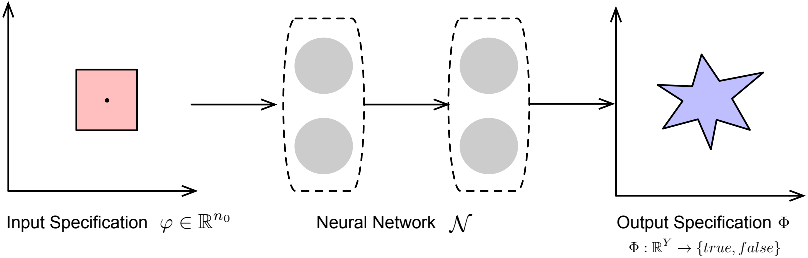
Figure 2: Within $\epsilon -$ **bounded** perturbated input should be classified as the similar result expected by humans.

A **well-trained robust** model can be viewed to maintain:

$$\forall \boldsymbol{x}, \hat{\boldsymbol{x}} \in \mathcal{I}, \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_p < \epsilon \Rightarrow \mathit{argmax}\ f_\theta(\hat{\boldsymbol{x}}) = \mathit{argmax}\ f_\theta(\boldsymbol{x})$$

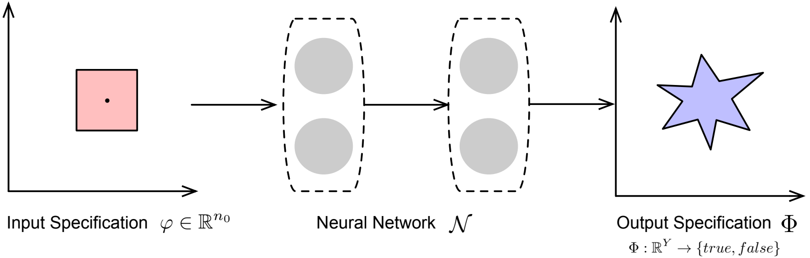
Neural Network Verification

Verification certify the network N with specification, in which the outputs of network (output specification Φ) hold the conditions of a connected region in the input domain (input specification φ)



Neural Network Verification

Verification certify the network N with specification, in which the outputs of network (output specification Φ) hold the conditions of a connected region in the input domain (input specification φ)



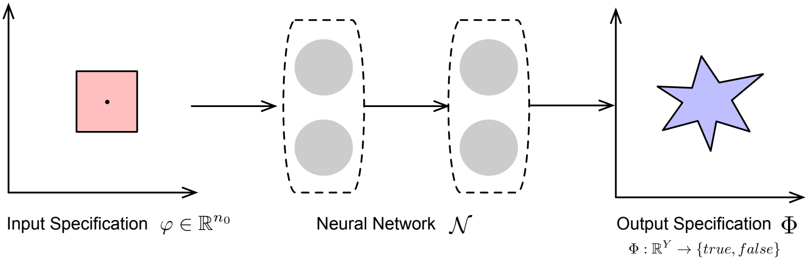
$$f_y(x+\sigma) \geq f_{y'}(x+\sigma), \;\; s.t., \forall y', \|\sigma\| \leq \epsilon$$

Probability of true class

Probability of other classes

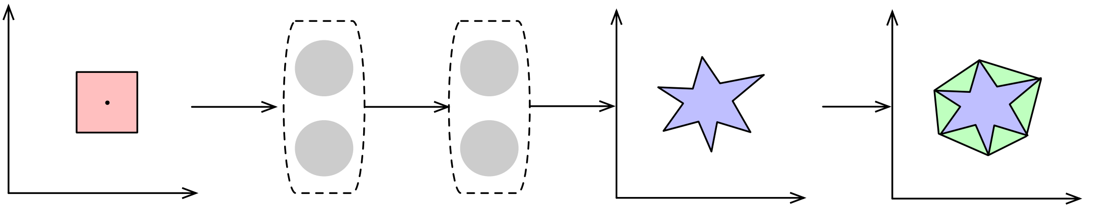
Perturbations of inputs

Neural Network Verification



Verifying a deep neural network from the reachable set of neural network output is **difficult**, because the output space is **complex**, non-convex set.

Neural Network Verification

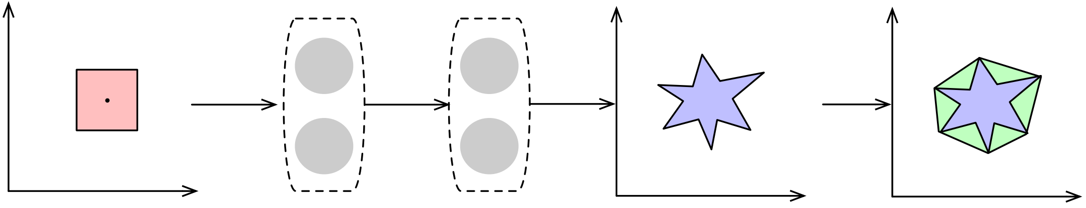


Verifying a deep neural network from the reachable set of neural network output is **difficult**, because the output space is **complex**, non-convex set.

Existing approaches:

- Complete Verification: **Exact** reachable set via combinatorial optimization (Reluplex).

Neural Network Verification

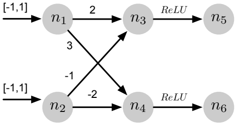


Verifying a deep neural network from the reachable set of neural network output is **difficult**, because the output space is **complex**, non-convex set.

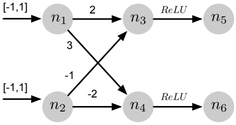
Existing approaches:

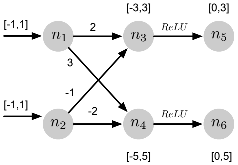
- Complete Verification: **Exact** reachable set via combinatorial optimization (Reluplex).
- Incomplete Verification: Optimize over an **over-approximation**.

Verification in Practice



Verification in Practice





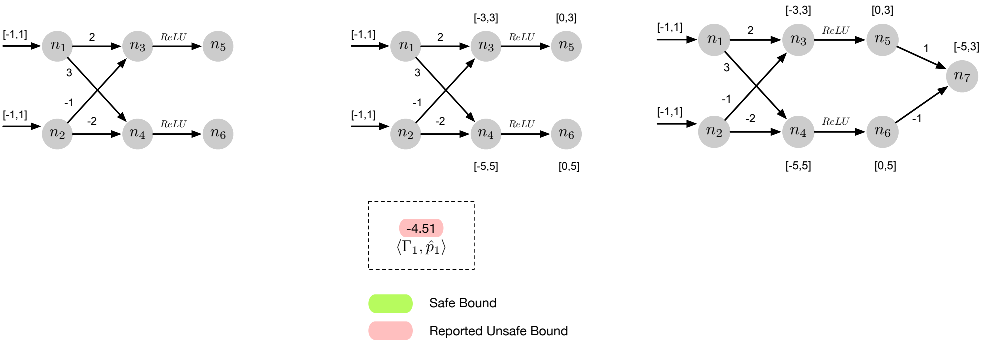
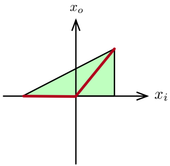
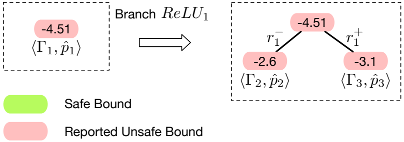


Figure 4: n_7 can be optimized lower bound as -4.51, which is a reported unsafe bound.

Template Generation from Branch and Bound



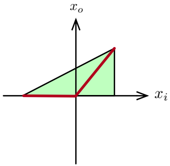
Approximated over ReLU



We add more constraints to the approximation:

$$\begin{aligned} ReLU_1 &\leq 0 : x_i < 0 \land x_o = 0 \\ ReLU_1 &\geq 0 : x_i > 0 \land x_o = 0 \end{aligned} \tag{1}$$

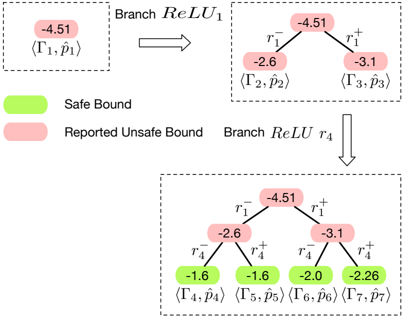
Template Generation from Branch and Bound



Approximated over ReLU

But we find it is still smaller than -2.5,
we have to branch more ReLU

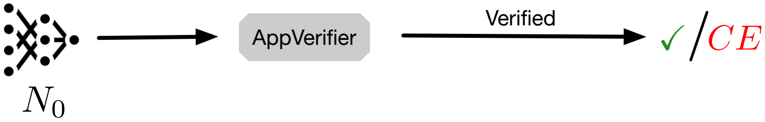
$$\begin{aligned} ReLU_4 &\leq 0 : x_i < 0 \wedge x_o = 0 \\ ReLU_4 &\geq 0 : x_i > 0 \wedge x_o = 0 \end{aligned} \tag{2}$$



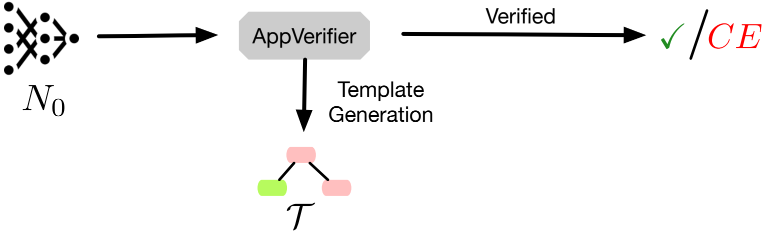
Successfully Verified the task, as all branches in subproblems are greater than -2.5.

Incremental Neural Network Verification

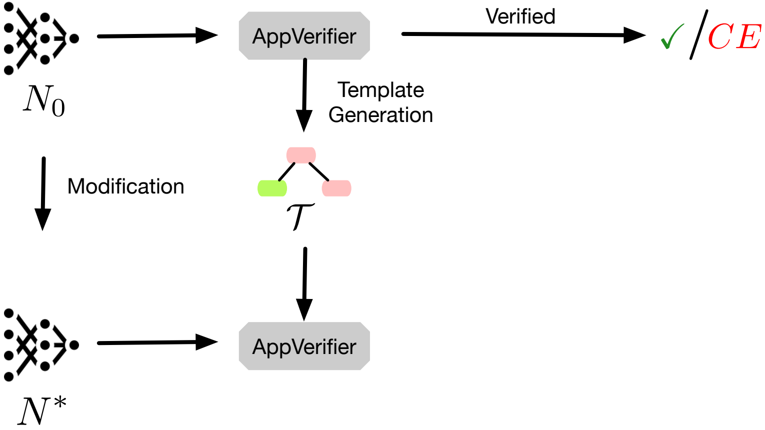
Existing Approach: IVAN



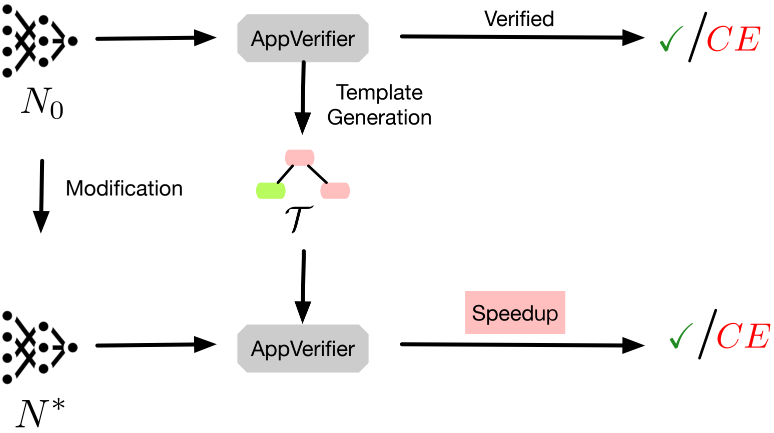
Incremental Neural Network Verification (cont.)



Incremental Neural Network Verification (cont.)



Incremental Neural Network Verification (cont.)



Noted: N* has the identical structure to N but slightly differs in model parameters.

Aim of This Work

- **Existing** work, such as IVAN, does not consider the order of the subproblem issues.
- We consider the counterexample potentiality of each node in the specification tree.
- We also consider the classic problem of *"exploration and exploitation" trade-off* in search-based techniques.

Contribution - Olive

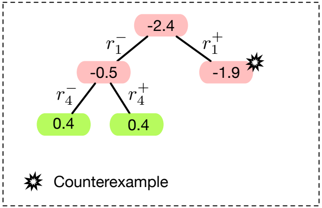
Order-**l**eading **i**ncremental **ve**rification (Olive) approach explores the reusing template tree, guided by **counterexample potentiality order** to termintate verification upon encountering a counterexample during the neural network verification process.

Contribution - Olive

Order-**l**eading **i**ncremental **ve**rification (Olive) approach explores the reusing template tree, guided by **counterexample potentiality order** to termintate verification upon encountering a counterexample during the neural network verification process. We proposed two versions of Olive:

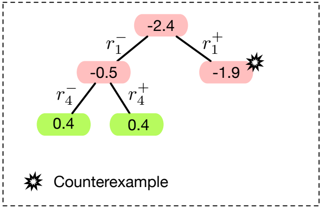
- Olive^g: A **greedy** strategy favors the exploitation of the tree nodes in which a **counterexample** is more likely to be found.
- Olive^b: a **balanced** strategy that also explores the tree nodes that seem less likely to contain counterexamples, in case there is a considerable gap between N and N^* and thus the template is not precise enough.

Order-leading Incremental Approach (1): *Olive*^{g}

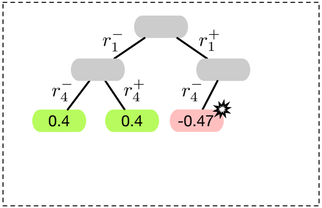


(a) BaB for N

Order-leading Incremental Approach (1): *Olive*^{g}

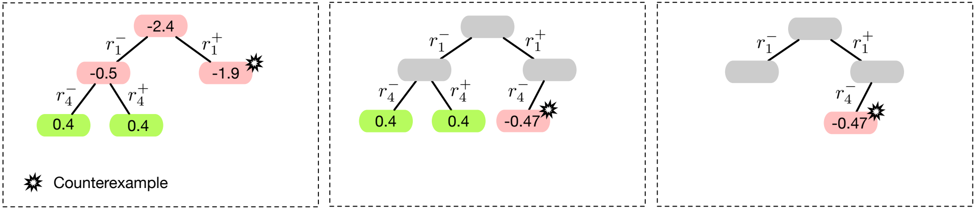


(a) BaB for N



(b) IVAN for N^*

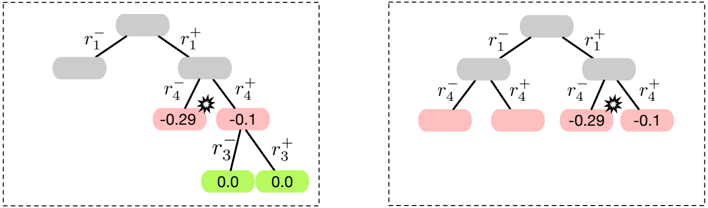
Order-leading Incremental Approach (1): *Olive^g*



(a) BaB for N (b) IVAN for N^* (c) *Olive^g* for N^*

Figure 5: Comparison of Olive^g with BaB and Ivan for verification of N^*

In Olive^g, the order $\sqsubset_G$ over the set S initially depends on the information from the specification tree of N. In each loop, it selects the node that is most likely to contain a counterexample, and expands its children if the approximated lower bound is negative and the counterexample is **spurious**. Since $\sqsubset_G$ **prioritizes** the child that has a greater depth, Olive^g favors exploiting all the descendants of a node, until either a **counterexample** is found or the node is verified.



(a) *Olive*^{g} for N^*

(b) *Olive*^{b} for N^*

Figure 6: Comparison between Olive^g and Olive^b for verification of N^*

While the greedy strategy Olive^g favors exploitation of the suspicious sub-problem suggested by the specification tree $\mathcal{T}_N$ of N, it can **fail** when the suggestion given by $\mathcal{T}_N$ is not precise for N^*. Namely, there could exist a different branch in $\mathcal{T}_{N^*}$ other than the one being exploited, in which counterexamples are easier to be found. This raises the classic problem of *"exploration and exploitation" trade-off* in search-based techniques. In light of this, we propose a balanced strategy that models the incremental verification problem as a *multi-armed bandit (MAB)* **slivkins2019introduction** problem.

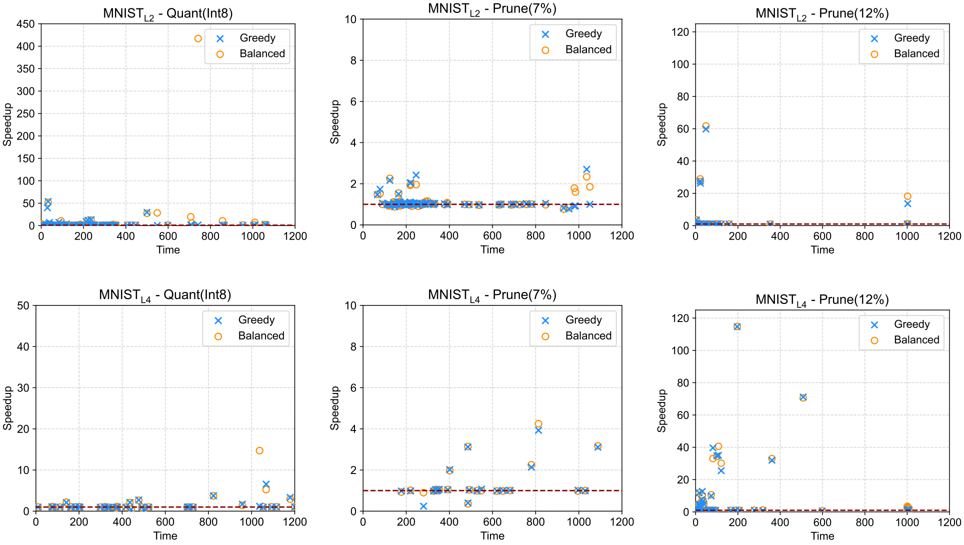
Table 1: The details of the benchmarks adopted in our experiments

Model (N)	**Architecture**	**Neurons**	**Dataset**	**Problem Instances**
$\texttt{MNIST}_{L2}$	$2\times$ 256 fully-connected layers	512	mnist	241
$\texttt{MNIST}_{L4}$	$4\times$ 256 fully-connected layers	1024	mnist	675
$\texttt{OVAL21}_{\texttt{BASE}}$	2 Conv, 2 fully-connected layers	4582	cifar	173
$\texttt{OVAL21}_{\texttt{WIDE}}$	2 Conv, 2 fully-connected layers	6244	cifar	207
$\texttt{OVAL21}_{\texttt{DEEP}}$	4 Conv, 2 fully-connected layers	6756	cifar	149

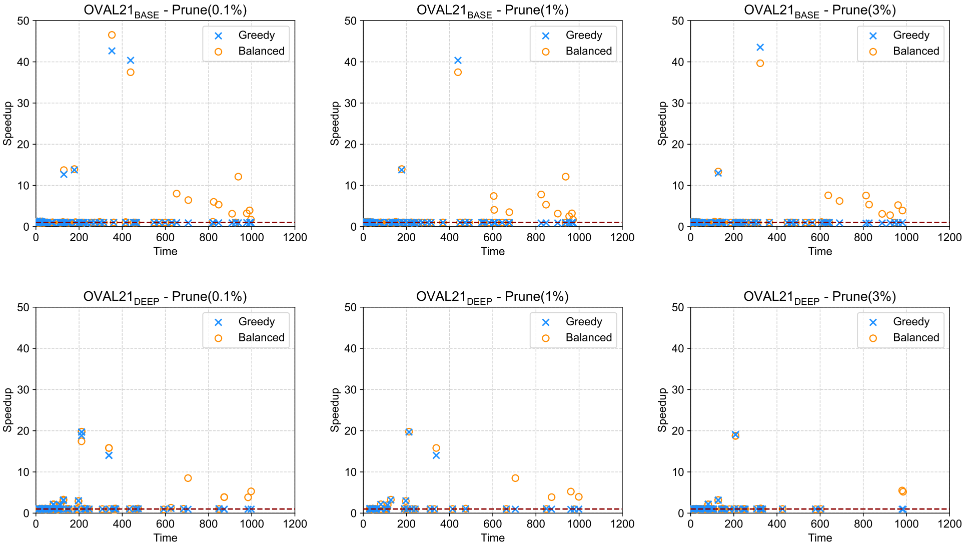
Table 2: The performance comparison between three incremental verification approaches and the BaB baseline approach, in terms of the average speedup w.r.t. BaB over the solved verification problems and the number of the additional problems that are not solved by BaB but solved by other approaches.

Model	Alteration	Ivan		Olive^g		Olive^b	
		Speedup	+Solved	Speedup	+Solved	Speedup	+Solved
$\mathtt{MNIST}_{\mathtt{L2}}$	Quant(Int8)	3.59×	8	3.28×	5	2.51×	3
$\mathtt{MNIST}_{\mathtt{L2}}$	Prune(7%)	1.60×	13	1.75×	10	1.73×	11
$\mathtt{MNIST}_{\mathtt{L2}}$	Prune(12%)	2.31×	0	34.89×	1	35.98×	4
$\mathtt{MNIST}_{\mathtt{L4}}$	Quant(Int8)	1.78×	4	2.15×	5	2.14×	6
$\mathtt{MNIST}_{\mathtt{L4}}$	Prune(7%)	1.59×	4	2.16×	2	2.18×	3
$\mathtt{MNIST}_{\mathtt{L4}}$	Prune(12%)	2.92×	2	14.74×	3	14.99×	23
$\mathtt{OVAL21}_{\mathtt{BASE}}$	Prune(0.1%)	1.74×	16	2.56×	11	2.58×	7
$\mathtt{OVAL21}_{\mathtt{BASE}}$	Prune(1%)	1.79×	17	2.37×	12	2.33×	7
$\mathtt{OVAL21}_{\mathtt{BASE}}$	Prune(3%)	1.77×	14	2.31×	10	2.29×	6
$\mathtt{OVAL21}_{\mathtt{DEEP}}$	Prune(0.1%)	1.91×	10	3.19×	8	3.21×	5
$\mathtt{OVAL21}_{\mathtt{DEEP}}$	Prune(1%)	1.92×	7	3.14×	5	3.18×	3
$\mathtt{OVAL21}_{\mathtt{DEEP}}$	Prune(3%)	1.88×	4	2.79×	2	2.78×	2
$\mathtt{OVAL21}_{\mathtt{WIDE}}$	Prune(0.1%)	1.98×	1	5.82×	1	5.98×	1
$\mathtt{OVAL21}_{\mathtt{WIDE}}$	Prune(1%)	1.74×	1	7.51×	1	7.34×	2
$\mathtt{OVAL21}_{\mathtt{WIDE}}$	Prune(3%)	1.88×	4	2.79×	2	2.78×	2

Experimental Evaluation 2-1



Experimental Evaluation 2-2



Questions

Thanks for listening, and welcomed with any questions.