ECE 584 Project Proposal: Efficient Incremental Verification for Neural Networks with Distinct Architectures

Lang Yin[*] Hung-Jui Chen[*]

Abstract

Verification via the branch-and-bound (BaB) method has been widely recognized and applied for neural network verification, and the latter is a subject on verifying whether a neural network satisfies a desired property. Nevertheless, constructing a verification tree even for one network could be computationally expensive, so a strategy for efficient verification across multiple networks has been desirable. Recent studies have proposed "incremental verification", a strategy which applies knowledge obtained from verifying an old network on new networks. Despite success, one of the major challenges in this topic arises when two neural networks have a distinct architecture, and this challenge has not been extensively studied. In this work, we plan to propose a mapping technique which identifies similar nodes between two networks, and as a guidance on the verification steps despite distinct architectures.

1. Introduction

Deep neural networks (DNNs) are achieving impressive results in many applications, from image recognition to autonomous driving. Despite these successes, guaranteeing DNN correctness is challenging, particularly for safety-critical tasks. Even small changes in inputs—or in the network's weights—can cause unforeseen misclassifications (Ugare et al., 2023a). Among these methods, *complete* verification methods based on branch-and-bound (BaB) address this challenge by exhaustively proving whether a DNN satisfies a given property for all inputs in a specified set (Shi et al., 2025). However, these methods often become infeasible when networks evolve (for example, through quantization or fine-tuning) and must be re-verified from scratch.

Incremental verification tackles this problem by reusing "proof artifacts" from an earlier verification run on the original network. In particular, branch-and-bound-based techniques produce a tree of subproblems (a "specification tree") that can be partially recycled when the network undergoes only small weight perturbations. The result is a large reduction in solver calls and branching steps, enabling much faster verification of updated models.

We will discuss more details in Section 3 about related work, but there are earlier works on this topic for networks with an **identical** architecture (Ugare et al., 2023a). However, in many scenarios, especially safety-critical domains like autonomous driving, upgrades may result in a differeent architecture. For example, as Figure 1 shows, upgrading a pedestrian detection model may change weights and slightly adjust the architecture (e.g., a hidden layer gaining a few neurons). Verifying the updated models both efficiently and correctly is crucial to maintain system safety without incurring high costs. The goal of this project is to investigate the possibilities of generalizing the earlier results for identical architecture to reliable methods for the case of distinct architectures.

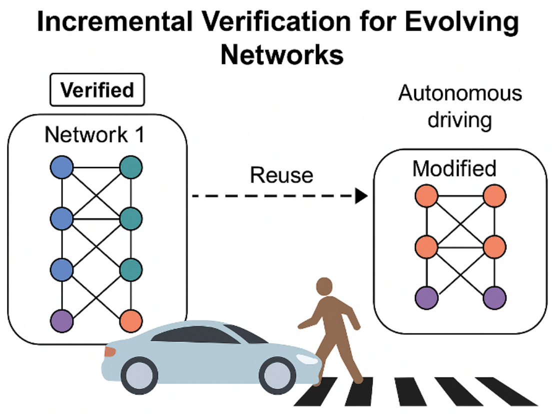


Figure 1. An example of architecture-shift after modifying the system.

[*]Equal contribution . Correspondence to: Lang Yin <langyin2@illinois.edu>, Hung-Jui Chen <hungjui2@illinois.edu>.

ECE 584 Class Project, Spring 2025. Instructor: Prof. Huan Zhang

2. Problem statement

Formally, consider a feedforward DNN

$$N \colon \mathbb{R}^{n_0} \to \mathbb{R}^{n_\ell}, \tag{1}$$

composed of ℓ layers and equipped with activation functions (e.g., ReLU). Let

$$q \subseteq \mathbb{R}^{n_0} \tag{2}$$

denote a (potentially high-dimensional) set of inputs of interest, and let

$$\varphi(\mathbf{z}) = (\boldsymbol{\alpha}^\top \mathbf{z} \geq 0) \tag{3}$$

be a specification on the output $\mathbf{z} \in \mathbb{R}^{n_\ell}$. The corresponding verification question asks whether

$$\forall \mathbf{x} \in q, \quad \varphi(N(\mathbf{x})) = \text{true}. \tag{4}$$

This is a *complete* decision problem: either φ is provably valid for every $\mathbf{x} \in q$, or there exists a concrete counterexample $\mathbf{x} \in q$ such that $\varphi(N(\mathbf{x}))$ is violated.

In *incremental* verification, we assume an *updated* network N' is produced by a small modification of the original N (for instance, adjusting weights via fine-tuning or applying quantization)(Ugare et al., 2023a). We then re-check

$$\forall \mathbf{x} \in q, \quad \varphi(N'(\mathbf{x})) = \text{true}, \tag{5}$$

but with the explicit goal of *reusing* the verification artifacts from proving φ on N. By leveraging prior branch-and-bound "split" decisions and partially certified subproblems, the incremental approach reduces redundant solver calls. Mathematically, this amounts to constructing and refining a so-called *specification tree*—capturing past branching—and adapting it to the new network N' (Palma et al., 2021). The resulting problem is both well-defined (the property φ and input domain q remain unchanged) and highly practical: it provides strict correctness guarantees with significantly reduced verification cost whenever N' differs modestly from N.

2.1. Distinct architectures

As discussed above, we focus on the case where two networks N and N' have distinct architectures. Since no earlier works have thoroughly studied in this case, we assume that the differences are modest at this stage. In particular, we assume:

- The networks N and N' have the same number of layers.
- Their layer types are identical. For instance, if Layer i in N is a linear combination (resp. ReLU) layer, then Layer i in N' is also a linear combination (resp. ReLU) layer.
- However, at least one layer other than the input and output layers have different numbers of nodes in N and N'.
- Perturbations on weights are expected among networks.

3. Related work

In addition to the most relevant recent work (Ugare et al., 2023a), various previous works have studied similar problems relevant to the general idea of reusing earlier results for new verifications. Those works include more efficient verifications (Palma et al., 2021; Fischer et al., 2022), cases with restrictions (Ugare et al., 2022; Wei and Liu, 2021), scalable verification (Liu et al., 2024), continuous verification (Ugare et al., 2023b), and more versatile verification protocols (Ferrari et al., 2022). However, to the best of our knowledge, all works in the area either do not study incremental verification, or even if they do, they only focus on reusing information obtained by verifying older networks with an identical architecture. As discussed earlier, our main focus is about the case with distinct architectures.

4. Methodology

4.1. Score function between neurons in two networks

At this moment, our proposed methodology centralizes at finding a mapping of nodes between the neurons in N and N'. The basic principle of this mapping is to match neurons with close output ranges. A natural approach is to compute ranges and match two neurons if the intersection of their ranges is the highest. But, with the ReLU function, a problem may occur. Consider this concrete example: Two neurons n and n_0 in the network N have ranges $I_n^N = [-100, 1]$ and $I_{n_0}^N = [-50, -25]$, and a neuron n' in the network N' has range $I_{n'}^{N'} = [-30, -20]$; these neurons lie in the same layer number in each of their respective networks, and are followed by the ReLU operation. The question is, which neuron should n' be mapped to? Clearly, $|I_{n'}^{N'} \cap I_n^N| = 10$ and $|I_{n'}^{N'} \cap I_{n_0}^N| = 5$. So if the mapping rule only cares about the intersection length, then n' should be mapped to n. But this match is problematic: The output of n' via the ReLU function must be zero, and so does the output of n_0, but it is not true for n, because n's range has a positive part. Therefore, this naive approach may not be optimal so we propose an alternative below.

Let $I = [a, b]$ and $J = [c, d]$ be two intervals in $\mathbb{R}$, and let X and Y be uniform random variables on the two intervals respectively. If the two intervals are matched, we would prefer a high likelihood of $X = Y$, i.e. $X - Y = 0$. Let $Z = X - Y$, clearly the probability density function of Z is supported only on $I \cup J \subsetneq \mathbb{R}$. By convolution, the probability density function is:

$$f_Z(z) = \int_{-\infty}^{+\infty} f_X(x) f_{-Y}(z-x)\mathrm{d}x \quad (6)$$
$$= \int_{I\cup J} f_X(x) f_{-Y}(z-x)\mathrm{d}x. \quad (7)$$

Therefore,

$$f_Z(0) = \int_{I\cup J} f_X(x) f_{-Y}(-x)\mathrm{d}x \quad (8)$$
$$= \int_{I\cup J} \frac{1}{(b-a)(d-c)}\mathrm{d}x \quad (9)$$
$$= \int_{I\cap J} \frac{1}{(b-a)(d-c)}\mathrm{d}x \quad (10)$$
$$= \frac{|I\cap J|}{|I|\cdot|J|}, \quad (11)$$

where the third equation holds because one of $f_X \cdot f_{-Y} \equiv 0$ on anywhere other than $I \cap J$. We call this value as the "score" function of I and J, written as $S(I, J)$ or $S(n, n_0)$ later for simplicity if I is the range of n and J is the range of n_0.

Let's revisit the example above. We have

$$S(n, n') = \frac{|I_{n'}^{N'} \cap I_n^N|}{|I_{n'}^{N'}|\cdot|I_n^N|} = \frac{10}{101\times 10} = \frac{1}{101};$$

and

$$S(n, n_0) = \frac{|I_{n'}^{N'} \cap I_{n_0}^N|}{|I_{n'}^{N'}|\cdot|I_{n_0}^N|} = \frac{5}{10\times 25} = \frac{1}{50}.$$

If the mapping rule prioritizes the score function, then n' is matched with n_0, as desired.

4.2. Mapping rule

Recall that for this work, we only consider two networks with the same number of layers and their types match, i.e. if Layer 3 is a ReLU layer in one network, then the same holds for the other. Therefore, our mapping rule is layer-by-layer: A neuron in Layer i will be mapped to a neuron in Layer i of the other network.

For each layer i, construct a complete bipartite graph G_i, where nodes in one part contains nodes representing the neurons in a layer of one network, and each edge has a weight, which is the score function between the two nodes. A successful mapping should match nodes with a high-weight edge.

Metaphorically, if two nodes are connected with a high-weight edge, then we may think the pair "wants" to be matched. This corresponds to the stable marriage problem, a classic problem in game theory introduced by Gale and Shapley in 1962 (Gale and Shapley, 1962). Formally, a matching is **stable** if there does not exist any pair of nodes from each part which both prefer each other to their current partner under the matching.

In the same paper, Gale and Shapley proposed an algorithm (the Gale-Shapley algorithm) solving this problem. We are going to apply that algorithm on our weighted bipartite graph here. Although the original Gale-Shapley algorithm only considered bipartite graphs with equal sides, it can be easily extended to unequal case: If part A has fewer nodes than part B, we may add $|B| - |A|$ "dummy" nodes to part A and connect them to all nodes in part B with weight zero. Once the algorithm returns a perfect matching M, we may remove the dummy nodes and the remaining matching is still stable although some nodes in B would be unmatched.

A rule of thumb is that every node in the second network N' must be matched with at least one node in the first network N, which has already been verified with a verification tree available. If, at a layer, N' has no more nodes than N, then this can be done by operating Gale-Shapley once. Otherwise, we develop the algorithm in the following steps, where A is the part for N for that layer and B is for N':

1. Run Gale-Shapley to match all nodes in A (with dummy nodes if necessary).

2. Delete nodes in B who were matched in the previous operation.

3. Repeat Step 1 until all nodes in B are matched.

4.3. Verification results update

Like Algorithm 3 in (Ugare et al., 2023a), we plan to update a node's status on the new verification tree (for the new network) according to the corresponding node's status on the old tree. In the case of identical architecture, the mapping is just the identical mapping.

5. Results

5.1. Experimental Setup

We evaluate incremental verification on MNIST using a fully–connected baseline network with two hidden layers of 256 neurons each ("256×2"). Three widened variants—260×2, 300×2, and 356×2—serve as target architectures. All local robustness properties are defined over an ℓ_∞–ball of radius $\varepsilon = 0.02$. We test two batch sizes, $k \in \{30, 50\}$ randomly sampled input images per architecture pair. The verifier employs LP relaxation, the RELU_ESIP split rule, and IVAN proof–transfer ($\alpha = \tau = 0.003$).

5.2. End–to–End Runtime

Table 1 reports the average speed-up obtained by incremental verification (IVAN) relative to a full, from-scratch run. Figure 2, 3 visualizes baseline and incremental run-times.

# Properties	Width change	Speed-up (×)
30	$256 \to 260$	**3.00**
30	$256 \to 300$	1.78
30	$256 \to 356$	1.76
50	$256 \to 260$	1.06
50	$256 \to 300$	1.04
50	$256 \to 356$	1.07

Table 1. Average speed-up (baseline / incremental) across architectures and property counts.

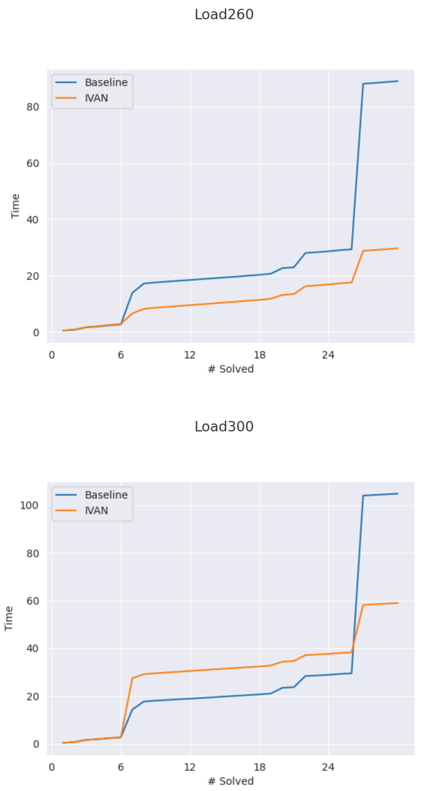


Figure 2. Average verification time per property: baseline vs. IVAN incremental(1).

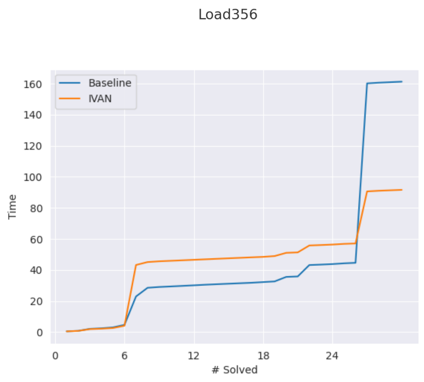


Figure 3. Average verification time per property: baseline vs. IVAN incremental(2).

5.3. Analysis

Incremental verification yields the largest benefit when the architectural difference is small: the 256→260 variant attains a **3**× reduction in runtime for $k = 30$ properties. As k grows to 50, the amortized mapping overhead reduces the net speed-up to roughly 1×. Wider gaps (256→300/356) still exhibit consistent gains of 1.7–1.8× for the smaller batch, suggesting that proof reuse remains valuable even with a 17% increase in layer width.

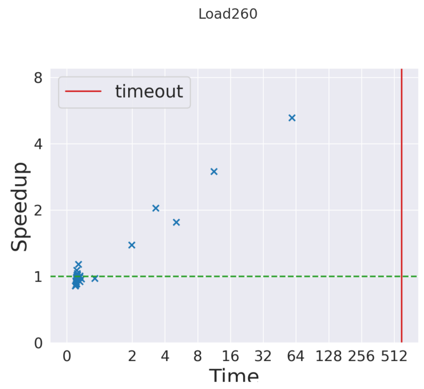
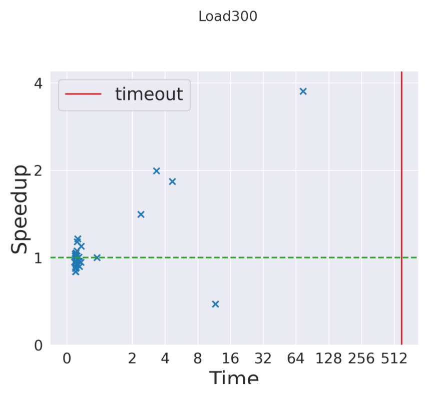


Figure 4. Speed-up vs. number of properties k, averaged over all three architecture pairs(1).



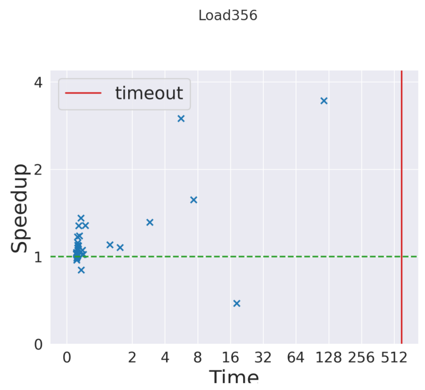


Figure 5. Speed-up vs. number of properties k, averaged over all three architecture pairs(2).

5.4. Take-away

Empirically, IVAN proof-transfer with Gale–Shapley neuron mapping can cut verification time by up to $3\times$ on networks that differ by as little as 4 neurons per layer. The benefit diminishes as the property batch grows, indicating that future work should explore incremental mapping amortization or parallel property verification to sustain high speed-ups at scale.

6. Conclusion and future work

In this project, we explored the possibility of generalizing previous results of neural network verification between networks with an identical architecture to the case where two networks share distinct architectures, and obtained some promising early results. Nevertheless, there are many directions for further directions. We discuss two potential improvements to what we already have now.

More diverse architectures. In this project, we consider networks with linear and ReLU layers only, and the architecture shifts among networks only involve a number of more or fewer neurons within a fixed number of layers. An ambitious direction is removing these constraints or at least applying more diverse constraints, such as introducing non-linear attention layers like transformers and considering the case where new networks have different numbers of layers.

Generalizing distributions. As mentioned, in this project, we consider the uniform distributions on the intervals. For future work, this assumption can be extended to normal distributions, which could be a more realistic setting. In this extended setting, let $X \sim N(\mu_X, \sigma_X^2)$ and $Y \sim N(\mu_Y, \sigma_Y^2)$ be Gaussian distributions on $I = [a, b]$ and $J = [c, d]$, where the values of mean and variance can be computed by a sufficient amount of empirical observations. For instance, given a data set $\{x_i\}_{i=1}^N$ of pre-activation values for a neuron in the first network, then we can compute

$$\mu_X = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \sigma_X^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_X)^2. \qquad (12)$$

Same computations can be applied on values for neurons in the second network. Therefore, we can easily compute the same quantities for $Z = X - Y$:

$$\mu_Z = \mu_X - \mu_Y, \quad \sigma_Z^2 = \sigma_X^2 - \sigma_Y^2, \qquad (13)$$

and by definition, the probability density function is

$$f_Z(z) = \frac{1}{\sqrt{2\pi(\sigma_X^2 + \sigma_Y^2)}} e^{-\frac{(z-(\mu_X-\mu_Y))^2}{2(\sigma_X^2+\sigma_Y^2)}}. \qquad (14)$$

Clearly,

$$f_Z(0) = \frac{1}{\sqrt{2\pi(\sigma_X^2 + \sigma_Y^2)}} e^{-\frac{(\mu_X-\mu_Y)^2}{2(\sigma_X^2+\sigma_Y^2)}}. \qquad (15)$$

Equations 12 and 15 provide computable methods to derive meaningful "scores" for pairs of neurons between networks.

References

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2022.

Marc Fischer, Christian Sprecher, Dimitar Iliev Dimitrov, Gagandeep Singh, and Martin Vechev. Shared certificates for&nbsp;neural network verification. In *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part I*, page 127–148, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-13184-4. doi: 10.1007/978-3-031-13185-1_7. URL `https://doi.org/10.1007/978-3-031-13185-1_7`.

D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. ISSN 00029890, 19300972. URL http://www.jstor.org/stable/2312726.

Jiaxiang Liu, Yunhan Xing, Xiaomu Shi, Fu Song, Zhiwu Xu, and Zhong Ming. Abstraction and refinement: Towards scalable and exact verification of neural networks. *ACM Trans. Softw. Eng. Methodol.*, 33(5), June 2024. ISSN 1049-331X. doi: 10.1145/3644387. URL https://doi.org/10.1145/3644387.

Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H. S. Torr, and M. Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition, 2021. URL https://arxiv.org/abs/2104.06718.

Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities, 2025. URL https://arxiv.org/abs/2405.21063.

Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Proof transfer for fast certification of multiple approximate neural networks. *Proc. ACM Program. Lang.*, 6 (OOPSLA1), April 2022. doi: 10.1145/3527319. URL https://doi.org/10.1145/3527319.

Shubham Ugare, Debangshu Banerjee, Sasa Misailovic, and Gagandeep Singh. Incremental verification of neural networks. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1920–1945, June 2023a. ISSN 2475-1421. doi: 10.1145/3591299. URL http://dx.doi.org/10.1145/3591299.

Shubham Ugare, Debangshu Banerjee, Tarun Suresh, Sasa Misailovic, and Gagandeep Singh. Toward continuous verification of dnns. *ICML WFVML Workshop*, 2023b.

Tianhao Wei and Changliu Liu. Online verification of deep neural networks under domain or weight shift. *RSS R4P Workshop*, 2021. URL https://arxiv.org/abs/2106.12732.