Proof Transfer for Fast Certification of Multiple Approximate Neural Networks

SHUBHAM UGARE, University of Illinois Urbana-Champaign, USA
GAGANDEEP SINGH, University of Illinois Urbana-Champaign and VMware Research, USA
SASA MISAILOVIC, University of Illinois Urbana-Champaign, USA

Developers of machine learning applications often apply post-training neural network optimizations, such as quantization and pruning, that approximate a neural network to speed up inference and reduce energy consumption, while maintaining high accuracy and robustness. Despite a recent surge in techniques for the robustness verification of neural networks, a major limitation of almost all state-of-the-art approaches is that the verification needs to be run from scratch every time the network is even slightly modified. Running precise end-to-end verification from scratch for every new network is expensive and impractical in many scenarios that use or compare multiple approximate network versions, and the robustness of all the networks needs to be verified efficiently.

We present FANC, the first general technique for transferring proofs between a given network and its multiple approximate versions without compromising verifier precision. To reuse the proofs obtained when verifying the original network, FANC generates a set of *templates* – connected symbolic shapes at intermediate layers of the original network – that capture the proof of the property to be verified. We present novel algorithms for generating and transforming templates that generalize to a broad range of approximate networks and reduce the verification cost. We present a comprehensive evaluation demonstrating the effectiveness of our approach. We consider a diverse set of networks obtained by applying popular approximation techniques such as quantization and pruning on fully-connected and convolutional architectures and verify their robustness against different adversarial attacks such as adversarial patches, L_0, rotation and brightening. Our results indicate that FANC can significantly speed up verification with state-of-the-art verifier, DeepZ by up to 4.1x.

CCS Concepts: • **Theory of computation → Program analysis**; **Abstraction**; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: Verification, Approximation, Robustness, Deep Neural Networks

ACM Reference Format:
Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2022. Proof Transfer for Fast Certification of Multiple Approximate Neural Networks. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 75 (April 2022), 29 pages. https://doi.org/10.1145/3527319

1 INTRODUCTION

Deep neural networks (DNNs) are being increasingly deployed for safety-critical applications in many domains including autonomous driving [Bojarski et al. 2016], healthcare [Amato et al. 2013], and aviation [Julian et al. 2018]. However, the black-box nature of the DNNs limit their trustworthy deployment. The common practice to evaluate the networks on a finite set of test inputs is insufficient to guarantee that the network will behave as expected on similar unseen

Authors' addresses: Shubham Ugare, University of Illinois Urbana-Champaign, USA; Gagandeep Singh, University of Illinois Urbana-Champaign and VMware Research, USA; Sasa Misailovic, University of Illinois Urbana-Champaign, USA.

2475-1421/2022/4-ART75
https://doi.org/10.1145/3527319

inputs. Researchers have shown that the state-of-the-art networks can be unreliable in the presence of an adversary, who can craft inputs that are similar to a correctly classified example but are misclassified by the DNN [Carlini and Wagner 2017; Goodfellow et al. 2015; Szegedy et al. 2014].

To tackle this issue, a growing line of work aims to prove the safety and robustness of deep neural networks in the presence of an adversary. For instance, ISO recently issued a standard for the assessment of the robustness of DNNs in safety-critical systems strongly emphasizing the role of verification [ISO 2021]. The verification approaches employ symbolic reasoning to prove that the network output does not misbehave on an infinite set of similar inputs. Most of the existing verification works reason about network's *local robustness* (or robustness, for short). Local robustness requires that all inputs in the neighborhood of a given input are classified to the same label. The guarantees that the existing verification techniques provide are either deterministic [Gehr et al. 2018; Katz et al. 2017, 2019; Lu and Kumar 2020; Salman et al. 2019b; Singh et al. 2018b, 2019b] or probabilistic [Bastani et al. 2016; Cohen et al. 2019; Lécuyer et al. 2019; Salman et al. 2019a; Weng et al. 2018b]. The main challenge that all these works try to address is minimizing the cost of running the verification algorithm while also achieving high *precision* – the fraction of local robustness properties that the verifier can prove.

Problem of Existing Work: Deploying DNNs on real-world systems has opened the question of optimizing the computational cost of inference: to reduce this cost, researchers have devised various techniques for *approximating* DNNs, which simplify the structure of the network (typically post network training), while maintaining high accuracy and robustness. Common approximation techniques include quantization (reducing numerical precision of the weights) [Gholami et al. 2021], pruning (removing weights or groups of weights) [Frankle and Carbin 2019], operator approximation (e.g. approximating convolutions [Figurnov et al. 2016; Sharif et al. 2021]) and others. Further, even after deployment, the developers may need to re-optimize the neural network if they detect a distribution shift [Rabanser et al. 2019].

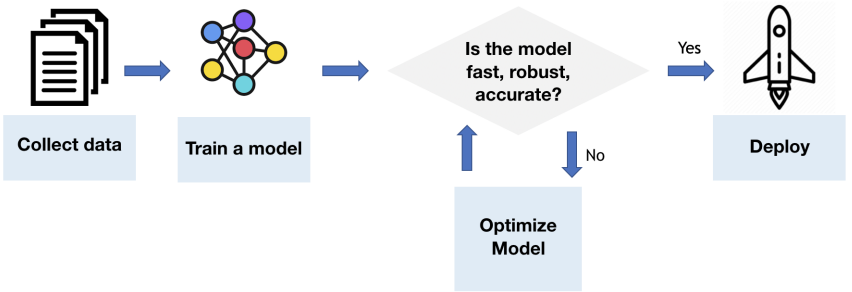


Fig. 1. A common deployment cycle for a deep neural network.

Figure 1 illustrates the DNN deployment process, which iteratively optimizes the network. Checking whether the optimized networks are robust is an important step in this process, but currently developers mostly rely on empirical testing due to the high cost of verification techniques. A major limitation of almost all existing approaches for verifying deep neural networks is that the verifier needs to be run from scratch end-to-end every time the network is even slightly modified. Running precise verification from scratch is an expensive operation and cannot keep up with the rate at which the networks are modified during deployment. Overcoming this main limitation requires addressing a fundamental problem in verifier design:

Can we reuse the proofs obtained when verifying a given network to speed up the verification of its multiple approximate versions?

This Work: This paper presents FANC, the first general approach for transferring proofs between a given network and its multiple approximate versions. Our approach is generic and can be combined with any existing state-of-the-art incomplete verifier [Gehr et al. 2018; Katz et al. 2017, 2019; Lu and Kumar 2020; Salman et al. 2019b; Singh et al. 2018b, 2019b] to improve the speed of verifying the approximate versions of the given network with fully-connected and convolutional layers and various activation functions. FANC guarantees to be as precise as the chosen verifier.

FANC first generates a set of *templates* – connected symbolic shapes (e.g., boxes, polyhedra) at an intermediate layer – by running a verifier through the original network. The templates capture the proof of the property to be verified on the original network. Next, FANC transfers these templates to the approximate networks by incremental template modification so that they capture the proof on the approximate versions. Finally, it runs the verifier on the approximate networks but only until the layer where the template is employed. If the intermediate proof is captured by the generated templates, then we have proved the property without the need to run the verifier end-to-end.

To make proof transfer efficient in practice our new algorithms for proof transfer across networks that we describe in Section 4 resolve the three main challenges: (1) Generating templates on the original network that are likely to be effective for verifying multiple approximate networks while also minimizing the cost of template generation; (2) Efficiently transforming the templates to further increase their effectiveness when verifying the approximate networks; (3) Enabling inexpensive checking if the generated template subsumes the intermediate proof on the approximate network.

Results: We thoroughly evaluate the effectiveness of FANC by verifying robustness of challenging fully-connected and convolutional networks approximated with quantization and pruning against different attacks. We considered four network architectures, robustly-trained on the popular MNIST and CIFAR10 datasets. We verified the robustness of the networks quantized using float16, int16, and int8 strategies against five different adversarial attacks: adversarial patches, L_0-random, L_0-center, rotation and brightening. We used the state-of-the-art DeepZ [Singh et al. 2018b] as the baseline verifier. FANC has significantly improved verification time for the quantized networks, by up to 4.1x, with a median speedup of 1.55x over DeepZ. We verified the robustness of of the networks with 10-90% pruning rates against the adversarial patch attack. FANC has improved verification time up to 2.8x, with a median speedup of 1.48x over DeepZ.

Contributions: Our main contributions are:

- We present a novel concept of proof transfer across a given DNN and its approximate versions.
- We characterize the class of practical approximate networks where our approach is applicable. This class encompasses networks obtained by quantization, pruning, or other transformations, with a bounded layer output difference.
- We instantiate our concept into a general practical framework that leverages new algorithms to significantly speed up existing verifiers without sacrificing their precision.
- We implement our approach into a tool named FANC and thoroughly evaluate the effectiveness of our approach for verifying networks generated by various types of approximation techniques on robustly-trained fully-connected and convolutional networks, and against different adversarial attacks. Our results indicate that FANC can significantly speed up verification over the baseline verifier, DeepZ.

FANC implementation is open-source, publicly available at https://github.com/uiuc-arc/FANC.

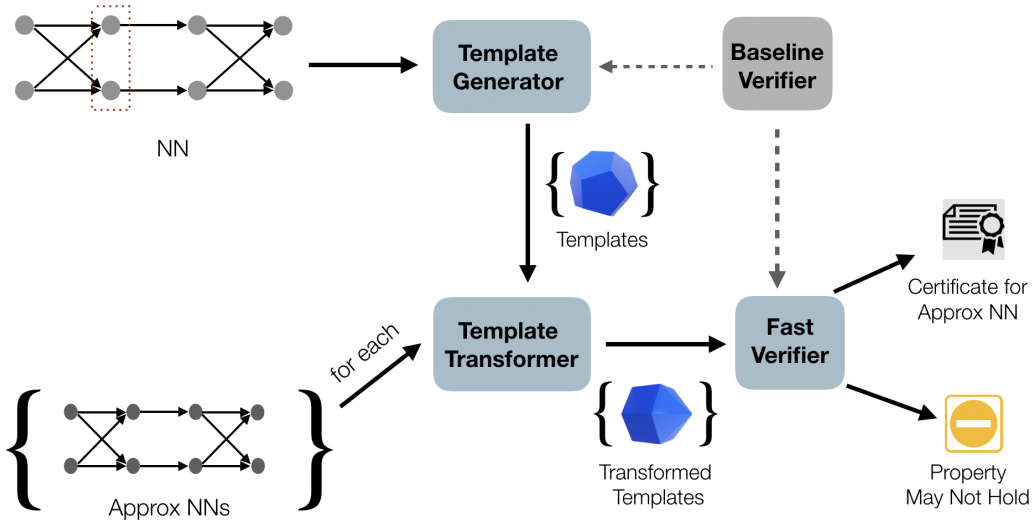


Fig. 2. Workflow of FANC from left to right. FANC consists of three components: template generator, template transformer, and fast verifier. First, the template generator takes the network N as input and creates a set of templates. For each approximate network, the template transformer transforms the templates. This transformed template is used by our fast verifier to verify the approximate network. The fast verifier either successfully verifies the network and generates a certificate or reports that the property may not hold.

2 OVERVIEW

Notations: We consider a feedforward (fully-connected or convolutional) neural network (NN) $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$ with l layers, n_0 input neurons and n_l output classes. The network classifies an input X to a class Y that has the largest corresponding output score, i.e. ,$Y = argmax_i[N(X)]_i$. We denote the partial neural network function between layers p and q, s.t. $p \leq q$ as $N_{p:q}$. Hence, we represent first k layers as $N_{1:k}$ and the remaining layers as $N_{k+1:l}$.

Workflow: Figure 2 illustrates the high-level idea of FANC. It takes as input a neural network N, its approximate version N^{app}, a set of input regions ϕ as precondition and the output property ψ as the postcondition. The neural network verification problem corresponding to the property (ϕ,ψ) involves proving that for all network inputs in ϕ, the corresponding network output satisfies the postcondition ψ. The goal of FANC is to speed up the proof of the property (ϕ, ψ) on the approximate network by leveraging intermediate proofs obtained by verifying N. FANC proves the same number of properties as a vanilla baseline verifier (in our case DeepZ) without templates but significantly faster. We get up to 4.1x speedup on robustness certification tasks (Section 6).

First, FANC creates a set of valid templates for the original network N via our template generation algorithm (Section 4.3), where each valid template T is a connected symbolic region defined only over the neurons of an intermediate network layer $k < l$ such that $N_{k+1:l}(T)$ satisfies ψ. These templates are used for speeding up the verification of the approximate network. The generated templates are next transferred to the same intermediate layer k of the approximate network N^{app} by transforming them via our template transformation algorithm (Section 4.4) which changes them to T^{app} such that $N^{app}_{k+1:l}(T^{app})$ satisfies ψ. This transfer is not possible between any two arbitrary networks but is facilitated in our setting because of the similarity between N and its approximate

version N^{app} which is essential so that the approximate networks have similar behavior as the original network. We identify the class of networks where FANC is applicable (Section 4.1). In the final step, FANC uses the set of transformed templates for early stopping of the verification of the property (ϕ,ψ) at the layer k of the approximate network.

We next show the workings of our approach on a toy example in detail. We show how the robustness proofs from a given network can be reused for speeding up the robustness verification of its multiple approximate versions. We consider the popular local neural network verification problem [Singh et al. 2018b, 2019b] where the input region ϕ is defined in the local neighborhood of a given input point X.

Neural Network Verifier: We consider a verifier based on abstract interpretation. Such verifiers have been shown to be effective for neural network verification achieving high precision and scalability. These verifiers work in the following steps. The verifier is built upon an analyzer parameterized by an *abstract domain*. The analyzer starts by *soundly* abstracting the input region into an abstract element representable in the chosen domain. Next, it performs a forward analysis by propagating the abstract shape through the network in a layerwise manner. At each intermediate layer (e.g., affine or ReLU), it computes an abstraction that contains all concrete outputs at the hidden layer with respect to the input region by applying layerwise abstract transformers. The final output of the analyzer contains an abstract shape containing all the concrete outputs. On this over-approximate region, the verifier next checks if the property is satisfied either exactly by calling a solver or via an abstract transformer. The verifiers obtained in this manner are sound by construction: if the verifier proves a property then it holds, however, they are not complete: the verifier may be unable to prove a property that actually holds due to its use of abstractions.

We denote the analyzer function as $A_{\mathcal{D}}$ that takes a region $I(x)$ as its input. Here, $\mathcal{D}$ denotes the domain of abstraction used by the analyzer. $A_{\mathcal{D}}(I(x), N_{1:k})$ denotes the abstract shape produced by the analyzer at layer k. $A_{\mathcal{D}}(T, N_{k+1:l})$ denotes the output abstract shape produced by the analyzer when it is given the input region T at layer k. We use γ_k to refer to a concretization function that computes the set of concrete values of the neurons in the intermediate layer k represented by the abstract shape $A_{\mathcal{D}}(I(x), N_{1:k})$. For ease of notation, we will refer to γ_k as γ for the rest of the paper.

Example Networks: As our example, we consider a fully-connected feedforward neural network N with the commonly used ReLU activation function and its two approximate versions N^a and N^b in Figure 3. FANC also works with other activation functions but in this example, we focus on ReLU as it is the most popular activation. For the purpose of demonstration, we obtain networks N^a and N^b by perturbing the edge weights N. We use two approximate networks for the example to show two different ways our template transformation algorithm can transfer proofs from the original network to its multiple approximate versions. The particular approximation examples are not intended to accelerate the NN inference, but to demonstrate the concept of proof transfer (we evaluate more practical quantization and pruning approximations in Section 6).

The neural networks (NNs) in Figure 3 have three layers: one input layer, one hidden, and one output layer. For simplicity of exposition of our ideas, we show the affine and the ReLU as two separate layers. All the layers contain two neurons each. The weights of the edges in the affine layer are coefficients of the weight matrix. These values are usually detailed floating point numbers but we use whole numbers for simplicity. In Figure 3, the weight on the edge connecting x_1 and x_3 in network N is 5. The corresponding weights for N^a and N^b are shown in blue and red respectively and have the values 4, and 6 respectively. The weight above and below the neurons in the affine layer indicates bias that is the same for all three networks. Hence, for neural network N, the value of x_3 is calculated as $5x_1 + 5x_2 + 0$. The first affine layer is followed by a ReLU activation layer. The last output layer is another affine layer.

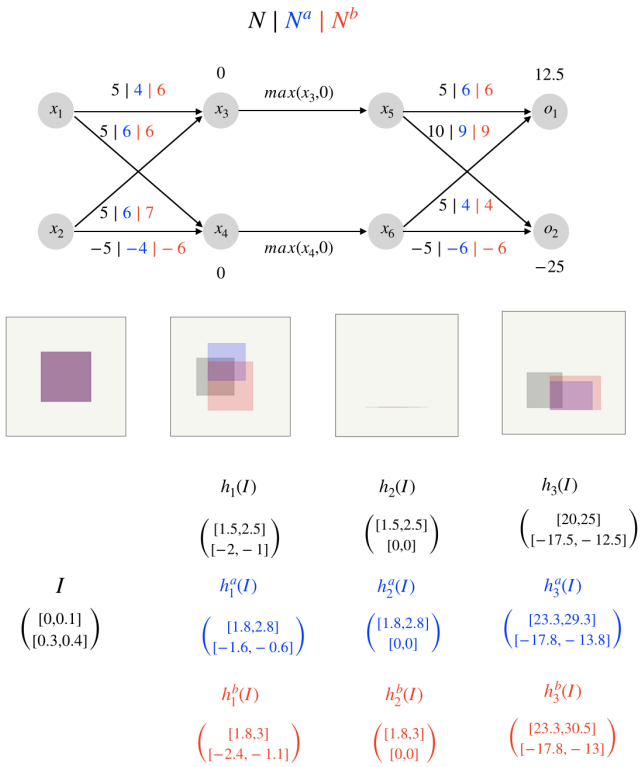


Fig. 3. Original network N and its approximations N^a (blue weights/shapes) and N^b (red weights/shapes).

Analysis with the Box Domain: In this example, we use the popular Box domain [Wang et al. 2018b] based verifier in FANC. However, FANC is generic and can be combined with other abstract domains such as Zonotopes [Singh et al. 2018b] or DeepPoly [Singh et al. 2019b]. The box analyzer propagates high dimensional intervals as abstract shapes. The input specification for our example is defined in terms of intervals of values that a particular pixel can take. Our key insight is that the abstract shapes produced by the analyzer for the original network N and the approximate networks N^a and N^b are similar as the networks are obtained by small perturbation of weights. Therefore, on a given input the output values at all layers for all of N, N^a, and N^b are close to each other. This closeness leads the approximate networks to have similar behavior as the original network. For the same reason, the abstract shapes over-approximating the concrete intermediate region produced by the analyzer are also similar (overlapping shapes in Figure 3). We use the notation $h_i(I)$ for $\gamma(A_{\mathcal{D}}(I, N_{1:i}))$, and similarly $h_i^a(I)$ and $h_i^b(I)$ for networks N^a and N^b. Based on this insight, we create a set of templates using N that enable us to perform faster verification of both N^a and N^b. We provide a metric to quantify the amount of closeness of the approximate network to the original one (see Section 4). This allows us to characterize the class of practical approximate networks where our approach is applicable.

In our example, we consider an input point $X = (0.05, 0.35)$ and an input specification around X, $I(X) = \begin{pmatrix} [0,0.1] \\ [0.3,0.4] \end{pmatrix}$. The output property ψ requires that the classifier should always classify all inputs in $I(X)$ to the label corresponding to o_1. Therefore $\psi := o_1 > o_2$. According to the

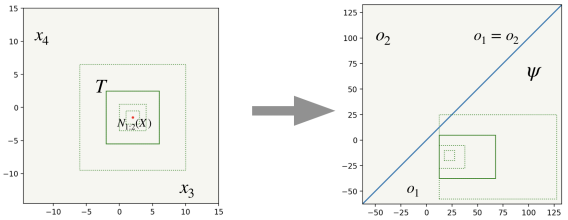


Fig. 4. Template generation for box abstraction. The box at intermediate layer is expanded until it violates the property at the output layer.

specification, the first input x_1 can take values in the range $[0, 0.1]$ and the second input x_2 can take values in the range $[0.3, 0.4]$. An affine layer transformer for the box analyzer will transform this box into another box over-approximating the effect of the affine layer on the input by using the standard addition and multiplication abstract transformers of the Box domain. Since, $x_3 = 5x_1 + 5x_2$ and $x_4 = 5x_1 - 5x_2$, hence the concretization γ of the transformer output is $\gamma(A_{\mathcal{D}}(I(X), N_{1:2})) = \{(x_3, x_4) | x_3 \in [1.5, 2.5], x_4 \in [-2, -1]\}$. Next, FANC handles the ReLU layer, which applied the ReLU function $f(X) = \max(0, X)$ on each input neuron. The box transformer for the ReLU layer transforms the interval for each neuron independently and for an input range $[lb, ub]$ the output is $[\max(lb, 0), \max(ub, 0)]$. For our running example, the input x_3 to the first ReLU lies in the range $[1.5, 2.5]$, and hence the box transformer returns $[1.5, 2.5]$ for x_5. The value for x_6 is computed similarly. The resulting box is similarly transformed at the next affine layer.

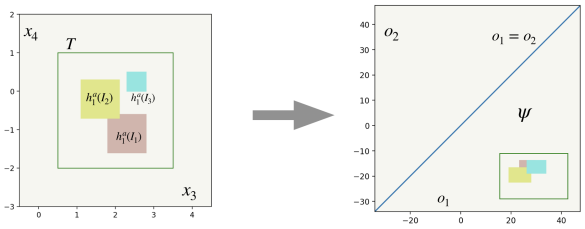
Proof Templates: For transferring the proof from N to N^a or N^b we create *proof templates* – connected symbolic regions that we create at an intermediate layer k having large volume. In the example, we create a template at the second layer i.e. $k = 2$. For FANC, we are more interested in templates that when propagated to the final layer map to a region where the property ψ holds. We call such a template a *valid template.* [1] Templates can make the verification more efficient: when T is a valid template then the containment of the abstract shape $A_{\mathcal{D}}(I, N_{1:2})$ in T is sufficient to conclude that N satisfies the property ψ for the input region I.

The main insight of FANC is that we can create a set of templates $\mathcal{T}$ for N and then transfer it to N^{app} as $\mathcal{T}^{app}$, where the transformation from $\mathcal{T}$ to $\mathcal{T}^{app}$ takes significantly lower time than the creation of $\mathcal{T}$ or $\mathcal{T}^{app}$ directly. $\mathcal{T}^{app}$ is a function of the particular approximate network. Our template transformer takes $\mathcal{T}$, the approximate network, and the original input X as the input for producing $\mathcal{T}^{app}$.

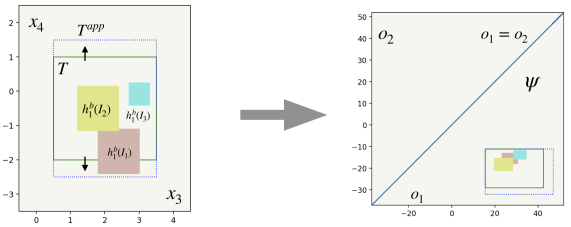
To generate a template around an input $X = (0.05, 0.35)$, we map X to the template layer by applying $N_{1:k}$, (Figure 4). We then create a box region containing $N_{1:k}(X)$. We expand this box as long as it is a valid template i.e. it verifies the property. Template T is valid for the network N by construction. However, it may not be valid for the networks N^a and N^b. Therefore, the template T is transformed by template transformer for a particular approximate network.

Proof Transfer to Approximate NN: Instead of expensive full verification, FANC creates a set of templates $\mathcal{T}^{app}$ such that the analysis on the approximate network propagates the input interval only till the template layer and checks if there exists a template $T \in \mathcal{T}^{app}$, that contains the abstract shape computed by the analysis. So we check if $h_1^a(I) \subseteq T$, if it is true then we do not need to propagate I further. If $h_1^a(I) \nsubseteq T$, we just propagate the shapes further to the output layer and proceed with the proof as without templates.

[1] We formally define a template in Section 3 and valid template in Section 4.2.



(a) Template and the box analyzer shapes at first layer and output layer for the neural network N^a.



(b) Template and the box analyzer shapes at first layer and output layer for the neural network N^b. The template T is transformed to T^{app}.

Fig. 5. Example showing template transformation for the approximate network

For our example, we consider three input regions for which we try to prove that ψ holds.

$$I_1 = \begin{pmatrix} [0, 0.1] \\ [0.3, 0.4] \end{pmatrix}, I_2 = \begin{pmatrix} [0.05, 0.15] \\ [0.15, 0.25] \end{pmatrix}, I_3 = \begin{pmatrix} [0.2, 0.25] \\ [0.25, 0.30] \end{pmatrix}$$

In our first example, for the verification of specifications (I_1, I_2 and I_3) on N^a, we skip the template transformation step and use $\mathcal{T}$ as $\mathcal{T}^{app}$.

Suppose we generated a template $T = \begin{pmatrix} [0.5,3.5] \\ [-2,1] \end{pmatrix}$ using the Template Generator on N. T must be valid for N by construction, but we need to check its validity for N^a. In Figure 5a we can see that the template is valid since the analyzer output for the template is mapped in the region $o1 > o2$. The next step after checking the validity of the templates is to verify the property for each input specification. Each input interval is propagated by the box analyzer to the second layer. The output $h^a(I_j)$ for each interval is shown in the figure. The template at the layer captures the propagated boxes for each input region i.e. $h^a(I_j) \subseteq T$. Now, we do not need to propagate I_1, I_2, and I_3 till the final layer. Overall, we invoked the analyzer to propagate I_1, I_2 and I_3 to the layer k, and invoked it one time to propagate T from layer $k+1$ to layer l to check the validity of T. This is computationally less expensive than propagating all I_1, I_2, and I_3 through all l layers of the network.

Template Transformation: In the next example, we show the need of template transformation for the proof on the network N^b. The templates created from the original networks may fail for more aggressive approximations and hence, we need to transform the templates to make them work on more practical networks. Again, we consider the same template $T = \begin{pmatrix} [0.5,3.5] \\ [-2,1] \end{pmatrix}$ from the previous example. First, we check the validity of T on N^b. Figure 5b shows the analyzer output for the template is mapped in the region $o1 > o2$, and hence it is valid on N^b as well. The next step is to check the containment of the intermediate shapes $h^b(I_j)$ for each specification I_j in the template. In the figure, we see that the $h_1^b(I_1)$ is not captured by the template T. FANC transforms the existing templates

in this step to solve this challenge. The transformation is applied even before checking the validity and the containment. Thus, there is no additional cost except the (small) transformation cost.

In our experiments, we observed that for a high dimensional template these intermediate shapes from the analyzer are captured along most dimensions but they fall outside the template along a few dimensions. In the example, as shown in Figure 5b, $h_1^b(I_2)$ and $h_1^b(I_3)$ are captured by the template, but $h_1^b(I_1)$ is not contained in the template along the x_4 dimension. One approach could be expanding the template again as in the template generation step, but a repeated validity check brings an unacceptable overhead. Instead, we use a faster approach that expands the template just along few dimensions that need fixing. We first compute $N_{1:2}^b(X)$ which is inexpensive as it does not require running the analyzer. The difference in $N_{1:2}(X)$ and $N_{1:2}^b(X)$ can be used to heuristically choose the dimensions along which we need to expand the template (see Section 4.4). In our example, a small expansion of T^{app} resulted by the expansion of T along the x_4 dimension fixes the problem. T^{app} captures all the intermediate shapes and hence, the proof transfer can be used for I_1 as well.

Time Complexity of FANC: For a network with l layers and maximum width (the largest number of neurons in a layer) n, and with the template captured at layer k ($k < l$), FANC's verification time per input specification with box as the baseline verifier is $O\left(\lambda k n^2 + (1-\lambda) l n^2\right)$. Here, the constant λ is the fraction of input specifications for which the intermediate abstract shape is captured by at least one valid template $T \in \mathcal{T}^{app}$. To maximize speedup, we want to create optimal valid templates of large volumes such that $\lambda \approx 1$. To reduce the overhead of template generation, FANC keeps the number of templates much smaller than the number of verification instances, and the templates are shared across multiple approximate networks to further amortize the costs. We do a more detailed time complexity analysis of FANC with arbitrary baseline verifier in Section 4.6

3 PRELIMINARIES

In this section, we formally provide the necessary background on neural network verification, the different types of adversarial attacks we verify the network to be robust against, the different approximations of neural networks considered in our work and the concept of proof transfer.

3.1 Neural Network Verification

Neural Networks: Neural networks are functions $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$. In this work, we focus on layered neural networks obtained by a composition of l layers $N_1 : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{N_1}, \ldots, N_l : \mathbb{R}^{n_{l-1}} \rightarrow \mathbb{R}^{n_l}$. Each layer N_i is either an affine layer (in particular, convolution with one or more filters is an affine transformation) or a non-linear activation layer like ReLU, sigmoid or tanh.

At a high level, neural network verification involves proving that all network outputs corresponding to a chosen set of input regions ϕ satisfy a given logical property ψ. Before providing further details, we first define the set of input regions that we consider in this work.

Definition 1 (Input Region). *For a neural network $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$, the set of **input regions** ϕ is such that $\forall I \in \phi$, I is a connected region and $I \subseteq \mathbb{R}^{n_0}$.*

In this paper, we focus on image classifiers N where with the pixel values in the image are in the range $[0, 1]$. We consider local input regions $I(X, \vec{\varepsilon}) \in \phi$ defined around a given input image $X \in \mathbb{R}^{n_0}$. The shape of $I(X, \vec{\varepsilon})$ is a high-dimensional box where each interval in the box indicates the range of values the corresponding pixel can take. $\vec{\varepsilon}$ is the vector of perturbation for each dimension. For convenience, we omit $\vec{\varepsilon}$ from the notation and use $I(X)$ as the input specification. We define the output property as:

Definition 2 (output specification). *For a neural network with n_l neurons in the output layer. $\psi : \mathbb{R}^{n_l} \rightarrow \{true, false\}$ is an **output specification** over the output region.*

For an output region $R \subseteq \mathbb{R}^{n_l}$ we use notation $\boxed{R \vdash \psi}$ meaning $\forall t \in R \,.\, \psi(t) = true$. For a NN, the neural network verification problem can be stated as:

Definition 3 (Local Verification). *The **local neural network verification** problem for a neural network N, an input region $I(X) \in \phi$ formed around the a given input X and a logical property ψ is to prove whether $\forall X' \in I(X).\ \psi(N(X')) = true$ or provide a counterexample otherwise.*

The output property ψ could be any logical statement taking a truth value true or false. In our paper, we focus on the property of classifier robustness.

Definition 4 (Robustness). *We say a neural network N on an input region $I(X) \in \phi$ **robustly** classifies to a label Y if $\forall X' \in I(X)$, $Y = argmax[N(X')]$.*

A verifier for the neural network N takes the set of input regions ϕ and a property ψ. For an input region $I \in \phi$, it returns a boolean true or false indicating if it can verify ψ for the region I or not. A complete verifier always verifies the property if it holds or returns a counterexample otherwise. We next formally define the soundness and completeness of verifiers:

Definition 5 (Sound Verifier). *A **sound verifier** V for an input region I, a neural network N, an output property ψ satisfies the following property:*

$$V(I, N) = true \implies \forall X \in I.\psi(N(X)) = true$$

Definition 6 (Complete Verifier). *A **complete verifier** V for an input region I, a neural network N, an output property ψ satisfies the following property:*

$$V(I, N) = true \iff \forall X \in I.\psi(N(X)) = true$$

The complete verifiers [Katz et al. 2017; Tjeng et al. 2019] are usually not scalable. An incomplete verifier is always sound: if it verifies the property, then the property holds but sacrifices completeness for scalability therefore it may fail to prove some properties that can be proven by a complete verifier. Most of the state-of-the-art incomplete verifiers [Gehr et al. 2018; Singh et al. 2019b] are based on abstract interpretation. These verifiers use an analyzer that performs a forward analysis by propagating abstract shapes from a chosen abstract domain $\mathcal{D}$ e.g., intervals [Wang et al. 2018b], octagons [Miné 2001], or polyhedra [Singh et al. 2017] through the different layers of the network. The abstract shapes at each layer over-approximate all possible concrete values with respect to the input region at that layer. Thus the output abstract shape represents an over-approximation of the concrete output values. If the property ψ is true for the over-approximating abstract shape, then the verifier can verify that the property holds over the given input region.

The analyzer function $A_{\mathcal{D}}$ takes two inputs, a region $R \subseteq \mathbb{R}^n$, and a function f such that the domain of f is $\mathbb{R}^n$. The output of the analyzer is an abstract element over-approximating the concrete values at that layer. We can get the concrete value from the abstract output of the analyzer using a concretization function γ.

Definition 7 (Layerwise Concretizing function). *For a network N with n_k neurons at layer k, the* layerwise concretization function *$\gamma_k : \mathcal{D} \rightarrow \mathbb{R}^{n_k}$ is defined as:*

$$\gamma_k(a) = \{v \mid v = \Pi_{n_k}(v'), v' = \gamma(a)\}$$

Here, the actual concretization function γ maps the abstract element to the concrete domain. Then the function Π_{n_k} projects the concrete region to the $\mathbb{R}^{n_k}$ which is over the space of output neurons at layer k. For the remaining paper, We abuse the notation and use γ as the concretization

function applied on a specific layer appropriately. For instance, if the analyzer output is at k'th layer then we assume that applying γ implies applying γ_k.

Hence, $\gamma(A_{\mathcal{D}}(I, N))$ is over-approximating the output at the output layer for N. A verifier returns true if for all the points in the output region obtained by concretizing the analyzer output at the output layer satisfies the property ψ. Hence, the verifier function $V(I, N) = \gamma(A_{\mathcal{D}}(I, N)) \vdash \psi$.

These abstract interpretation-based neural networks are sound and use sound analyzers.

A sound verifier based on abstract interpretation uses a sound analyzer. We define it as:

Definition 8 (Sound Analyzer). *A **sound analyzer** $A_{\mathcal{D}}$ for a region R, a function f satisfies the following property:*

$$\forall t \in R. f(t) \in \gamma(A_{\mathcal{D}}(R, f))$$

Here, $\gamma(A_{\mathcal{D}}(R, f))$ is the over-approximation of all the values output of f can take for inputs in R. Our concept of proof transfer is independent of a specific verifier, and it can be used complementary to any sound verifier using abstract interpretation.

We define the input regions $I(X)$ to capture regions where adversarial attacks [Szegedy et al. 2014] on the neural network can happen. Next, we describe the types of adversarial regions we consider.

3.2 Adversarial Regions

We consider three common adversarial attacks in this paper. Each type of attack defines a different kind of interval input region around a particular image. We consider that $n_0 = h \times w$, and thus, the input image $X \in [0, 1]^{h \times w}$.

3.2.1 Patch Perturbation. Here the attacker can arbitrarily change any $p \times p$ continuous patch of the image. The adversarial region $I^{i,j}_{p\times p}(X)$ defined by a patch over pixel positions $([i, i + p - 1] \times [j, j + p - 1])$ where $i \in \{1, \ldots h - p + 1\}, j \in \{1, \ldots w - p + 1\}$ is given by:

$$I^{i,j}_{p\times p}(X) = \{X' \in [0, 1]^{h\times w} \mid \forall (k, l) \in \pi^{i,j}_{p\times p}. X'_{k,l} = X_{k,l}\}$$

where

$$\pi^{i,j}_{p\times p} = \left\{(k, l) \;\middle|\; \begin{matrix} k \in 1, \ldots, h \\ l \in 1, \ldots, w \end{matrix}\right\} \setminus \left\{(k, l) \;\middle|\; \begin{matrix} k \in i, \ldots, i+p-1 \\ l \in j, \ldots, j+p-1 \end{matrix}\right\} \tag{1}$$

To prove a network is robust against all the possible patch perturbations, we need to consider all the possible input regions. For example, with $p = 2$, we need to verify 729 different regions for an MNIST classifier. The set of input regions for a single image can be defined as

$$\mathcal{I}_{p\times p}(X) = \bigcup_{\substack{i \in \{1, \ldots h - p + 1\} \\ j \in \{1, \ldots w - p + 1\}}} I^{i,j}_{p\times p}$$

3.2.2 L_0 Perturbation. In this attack model, an attacker can change any c (need not be continuous) pixels in an image arbitrarily. We define the input regions corresponding to these perturbations as

$$\mathcal{I}_c(X) = \{Y \mid \|Y - X\|_0 \leq c\}$$

To prove that a network is robust in this input region we consider all possible sets of c perturbed pixels separately. Let PS be such a set of pixels and $|PS| = c$. We define the input region for a particular set PS as

$$I_{PS}(X) = \{Y \in [0, 1]^{h\times w} \mid \forall (k, l) \notin S. Y_{k,l} = X_{k,l}\}$$

3.2.3 Rotation. Rotation attack model [Balunovic et al. 2019] considers the set of images obtained by rotating an image by an angle γ from the range of interval Γ. The rotation R_γ is applied to the input image for $\gamma \in \Gamma$ to obtain the perturbed image. The input region corresponding to the perturbation is defined as

$$\mathcal{I}_\Gamma(X) = \{Y \mid Y = R_\gamma(X), \gamma \in \Gamma\}$$

The input region for rotation is non-linear and has no closed-form solution. Thus, state-of-the-art approaches [Balunovic et al. 2019] verify the over-approximation of the input region.

3.2.4 Brightness. Next, we consider the brightness attack proposed in [Pei et al. 2017] for defining our adversarial regions. The input regions for the attack is defined as:

$$I_\delta(X) = \{Y \in [0,1]^{h \times w} \mid \forall i \in \{1, \ldots h\}, j \in \{1, \ldots w\}. 1 - \delta \leq X_{i,j} \leq Y_{i,j} \leq 1 \ \vee \ Y_{i,j} = X_{i,j}\}$$

3.3 Neural Network Approximations

Various NN approximation techniques [Blalock et al. 2020; Jacob et al. 2018] are used to compress the model size at the time of deployment, to allow inference speedup and energy-savings without significant accuracy loss. While our technique can work on most of these approximations. We discuss the supported approximations in Section 4.1. For the evaluation, we focus on quantization and pruning as these are the most common ones.

3.3.1 Quantization. Model quantization [Jacob et al. 2018] allows reduction in memory requirement and inference time by performing computations and storing tensors at lower bitwidths than the usual floating-point precision.

3.3.2 Pruning. Pruning techniques [Blalock et al. 2020] systematically remove weights from an existing network without losing significant accuracy. In general, pruning reduces the accuracy of the network, so it is trained further (known as fine-tuning) to recover some of the lost precision. The process of pruning and fine-tuning can be iterated several times, gradually reducing the network's size. This type of pruning is called iterative pruning.

3.4 Proof Sharing with Templates

Sprecher et al. [2021] introduced the concept of sharing proof between two different specifications defined around the same input to improve the overall verification time.

Definition 9 (Template). *For a neural network N, a verifier V based on an abstract domain $\mathcal{D}$, a $T_k \subseteq \mathbb{R}^{n_k}$ is a **template** at layer k, where n_k is the number of neurons at layer k.*

We use the same notion of templates for proof transfer between networks.

4 PROOF TRANSFER TO THE APPROXIMATE NETWORK

Algorithm 1 presents FANC's main algorithm for transferring a proof from a given network to its multiple approximations. It takes as inputs the original network N, a set of approximations of N, a set of local input regions ϕ around X, an output property ψ, and an analyzer $A_\mathcal{D}$ parameterized by an abstract domain $\mathcal{D}$.

First, on an input X, FANC creates templates using the original network and the analyzer $A_\mathcal{D}$ (Section 4.3). Then it loops over each approximate network $N^{app} \in \mathcal{N}^{app}$. For each iteration, it transforms these templates for the specific approximate network (Section 4.4). Finally, it verifies the approximate network using the transformed templates (Section 4.5). The algorithm returns a mapping B from each approximate network to a Boolean true or false. FANC is an incomplete verifier therefore if the output is true, then that means FANC could verify that particular specification (ϕ,ψ). Otherwise, the result is unknown.

Algorithm 1 Proof transfer algorithm

Input: N, set of approximate networks $\mathcal{N}^{app}$, input X, analyzer $A_{\mathcal{D}}$, property (ϕ, ψ)
Output: A mapping B from each approximate network to a Boolean indicating if the property is verified for that network.

1: $\mathcal{T}$ = generate_templates$(A_{\mathcal{D}}, N, X, \psi)$
2: **for** $N^{app} \in \mathcal{N}^{app}$ **do**
3: $\quad \mathcal{T}^{app}$ = transform_templates_approx$(X, \mathcal{T}, N^{app})$
4: $\quad B[N^{app}]$ = verify_approximate$(A_{\mathcal{D}}, N^{app}, \mathcal{T}^{app}, \phi, \psi)$
5: **end for**
6: **return** B

Next, we define the class of approximations to which FANC is applicable.

4.1 Class of Approximate Networks

We consider the set $\mathcal{N}$ of all networks that have the same architecture as N. From this set of networks, we define the set of similar approximate networks by using a parameter $c \in \mathbb{R}$ that bounds the difference between N and every other network N^{app} in the set at each layer on all inputs from a given set S (e.g., a subset of the test set). We define the output distance $\delta^k_{N,N^{app}}(X)$ between two networks at a layer k of the network:

$$\delta^k_{N,N^{app}}(X) = \frac{\|N_{1:k}(X) - N^{app}_{1:k}(X)\|}{\|N_{1:k}(X)\|} \tag{2}$$

We assume $\|N_{1:k}(X)\| > 0$ in the definition and only define $\delta^k_{N,N^{app}}(X)$ for such X. In practice, $\|N_{1:k}(X)\| = 0$ would be rare on a real input image. Using the above definition, we define the set of approximate networks generated from N that are parameterized by the constant c and function of the input set S:

$$\mathcal{N}_{c,N} = \{N^{app} \in \mathcal{N} \mid \sup_{\substack{X \in S \\ k \in \{1,\ldots l\}}} \delta^k_{N,N^{app}}(X) \leq c\} \tag{3}$$

We fix the dataset S for the problem and omit it from the notation $\mathcal{N}_{c,N}$. We do not fix a specific norm function for these definitions, however, for the experiments we use L_∞ norm (Section 6). We compute $\hat{c}$ as the left hand side of the inequality in Equation 3 i.e. $\hat{c} = \sup_{\substack{X \in S \\ k \in \{1,\ldots l\}}} \delta^k_{N,N^{app}}(X)$. In the evaluation we compute $\hat{c}$ for each approximate network (Section 6). For any $N^{app} \in \mathcal{N}_{c,N}$ and $X \in S$, since $\frac{\|N_{1:k}(X) - N^{app}_{1:k}(X)\|}{\|N_{1:k}(X)\|}$ is bounded by c (for small value of c), we expect $\gamma(A_{\mathcal{D}}(I(X), N^{app}_{1:k}))$ and $\gamma(A_{\mathcal{D}}(I(X), N_{1:k}))$ to be close to each other (this hypothesis is validated by our experiments). Hence, we can create templates $\mathcal{T}$ from N and transfer them to N^{app} with modifications along relatively fewer dimensions. For each N^{app} considered for verification, we transform the templates according to our transformation algorithm and create a set of templates $\mathcal{T}^{app}$.

4.2 Proof Templates for Neural Networks

We need to create templates that satisfy certain properties for them to be useful for accelerating verification. Next, we will describe these properties. First, we define the validity of a template.

Definition 10 (Valid Template). *A template T is a **valid template** for a neural network N, an analyzer $A_{\mathcal{D}}$, a property ψ and a layer k in the network if*

$$\gamma(A_{\mathcal{D}}(T, N_{k+1:l})) \vdash \psi$$

The condition states that the region obtained after concretization of the analyzer output satisfies the property ψ (Section 3.1). In this paper, we fix our analyzer $A_{\mathcal{D}}$ and the property ψ, hence we only talk about the validity of a template as function of layer k and the network N.

Our goal is to create templates that are valid for any approximation $N^{app} \in \mathcal{N}_{c,N}$. Additionally, we desire that the abstract shapes obtained at the intermediate layers are captured by them. The number of valid templates for the networks in $\mathcal{N}_{c,N}$ with respect to the chosen $A_{\mathcal{D}}$ are infinite. The problem of finding an optimal (in terms of volume) valid template requires computing the best abstraction in $\mathcal{D}$ that maximizes the volume and such that its image at the output layer propagated by $A_{\mathcal{D}}$ satisfies ψ. This is computationally expensive for domains like Box and not possible for other popular domains for neural network verification such as Zonotopes and DeepPoly that have no best abstraction.

Nevertheless, one thing we desire is to get large (in terms of volume) valid templates, since intuitively they will capture more intermediate shapes. Next, we formally define the notion of maximal valid templates that could be used for proof transfer and are more practical to compute.

Definition 11 (Maximal Valid Template). *T_M is a **maximal valid template** for a neural network N, an analyzer $A_{\mathcal{D}}$, a property ψ and a layer k in the network if*

$$\forall T \,.\, T_M \subseteq T \wedge \gamma(A_{\mathcal{D}}(T, N_{k+1:l})) \vdash \psi \implies T = T_M$$

Theorem 1. *Given a neural network N, a verifier V based on a sound analyzer $A_{\mathcal{D}}$, a property ψ and a valid template $T \subseteq T_M$ where T_M is a maximal valid template then $\forall t \in T.\psi(N_{k+1:n}(t)) = true$*

Proof. Since T_M is a valid template, $\gamma(A_{\mathcal{D}}(T_M, N_{k+1,n})) \vdash \psi$ and since V is a sound verifier, $\forall t \in T_M.\psi(N_{k+1:n}(t)) = true$. Given that $T \subseteq T_M$, we conclude that $\forall t \in T.\psi(N_{k+1:n}(t)) = true$. □

For verifiers based on monotonic analysis such as Box, we can prove an even stronger condition that $T \subseteq T_M$, then T is a valid template. However, this statement may not hold for verifiers based on non-monotonic analyzers, e.g., DeepZ [Singh et al. 2018b].

Definition 12 (Monotonic analyzer). *A **monotonic analyzer** $A_{\mathcal{D}}$ for regions R_1, R_2, and a function f satisfies the following property:*

$$R_1 \subseteq R_2 \implies \gamma(A_{\mathcal{D}}(R_1, N)) \subseteq \gamma(A_{\mathcal{D}}(R_2, N))$$

4.3 Template Generation Algorithm

Algorithm 2 presents our algorithm for template generation. As mentioned in Section 2, it is essential to minimize the cost of template generation for maximizing speedups. Generating a maximal valid template is computationally expensive even for simpler domains such as Box. Hence, we provide an algorithm that computes an under-approximation of the maximal valid template that works well in practice. We observed that even for perturbations that are semantically different, the network transformations successively map them onto more and more similar regions at the intermediate layers. Thus, the amount of overlap increases with the layers. L_∞ modifies all the pixels, patches modify small patch of adjacent pixels, whereas the pixels modified by L_0 and brightness may not be connected. We empirically observe that the templates based on L_∞ perturbations are effective to capture the proofs of other perturbations.

Our algorithm overcomes the challenge of finding the effective templates for the input X by splitting the input space. We uniformly split the image into continuous patches of size $\kappa \times \kappa$. For each such patch, we define an input region $I^{p_h,p_w}_{\kappa\times\kappa}(X, \epsilon)$, in which the pixels corresponding to the patch can be perturbed by at most ϵ. In our algorithm, we find the maximum ϵ such that the verifier

Algorithm 2 Template generation algorithm

Input: N, ψ, template parameter κ, intermediate layer k, input X
Output: Set $\mathcal{T}$ of templates

1: $\mathcal{T} = \{\}$
2: $H_{itr} = \frac{h}{\kappa},\ W_{itr} = \frac{w}{\kappa}$
3: **for** $i \in 1, \ldots H_{itr}$ **do**
4: **for** $j \in 1, \ldots W_{itr}$ **do**
5: $p_h = \kappa * i,\ p_w = \kappa * j$
6: $\epsilon_{max} = \max\{\epsilon \in \mathbb{R} \mid V(I^{p_h,p_w}_{\kappa\times\kappa}(X,\epsilon), N) = true\}$
7: $a_T = A_{\mathcal{D}}(I^{p_h,p_w}_{\kappa\times\kappa}(X,\epsilon_{max}), N_{1:k})$
8: $T = \gamma(\Pi_{box}(a_T))$
9: $\mathcal{T} = \mathcal{T} \cup \{T\}$
10: **end for**
11: **end for**
12: **return** $\mathcal{T}$

proves that the property ψ holds on the generated region. More formally, for the start height and width coordinates (p_h, p_w) and κ as the *patch size*, we define this region as:

$$I^{p_h,p_w}_{p\times p}(X,\epsilon) = \{X' \in [0,1]^{h\times w} \mid \|X' - X\|_\infty \leq \epsilon \wedge \forall (k,l) \in \pi^{p_h,p_w}_{p\times p}.\ X'_{k,l} = X_{k,l}\}$$

where $\pi^{p_h,p_w}_{p\times p}$ is the set of pixel indices not in the patch (Equation 1).

Line 4 iterate over the input's patches. For each patch, line 6 searches for the maximum ϵ such that the input patch $I^{p_h,p_w}_{\kappa\times\kappa}(x,\epsilon)$ is verified. i.e.

$$V\left(I^{p_h,p_w}_{\kappa\times\kappa}(X,\epsilon), N\right) = true$$

Line 7 runs the analyzer on the input region defined by ϵ_{max} till the k-th layer of N. Line 8 performs the conversion of the resulting abstract shape into box abstraction and then concretizes it. In practice, we perform our analysis on more precise domains than Box such as Zonotopes, we store the templates as box regions - since the containment check(Algorithm 1) is faster when the templates are boxes.

If the analyzer $A_{\mathcal{D}}$ is monotonic, a binary search (in line 6) can find the optimal ϵ. However, we found that even a less tight ϵ works well in practice. The DeepZ verifier that we used for our experiments is not monotonic, but we find a reasonable ϵ by searching through the set of values $\frac{1}{2^i}$ for a chosen integer i. Empirically, we observe that using a slightly smaller epsilon than the optimal epsilon can often result in templates that are more reusable across networks. Since the number of templates is smaller than the number of verification instances, the cost for template creation is smaller than verifying all the specifications. In Section 4.6, we will derive the time complexity of FANC showing its relation with the ratio of number of templates to the number of specifications.

4.4 Template Transformation

Template transformation is a key step in FANC. The templates created on the original network are not necessarily valid on approximate networks. They are valid only when the value of the constant c is small. For a broader set of networks with larger c, we need to transform the templates so that they are more general and therefore applicable to more networks. While the template is created only once for the original network and a given input X, we may perform template transformation

for multiple network approximations. The main challenge is in minimizing the transformation time while maximizing the generalization of the templates simultaneously.

For template generation, FANC requires propagating abstract shapes via the analyzer to an intermediate layer. However, as discussed in Section 2, running an expensive analyzer for the transformation step will offset any gains from using the templates. Hence, instead of propagating the input interval to the intermediate layer via an analyzer, we propagate the point X, i.e., we calculate $N^{app}_{1:k}(X)$, and then widen the original templates based on this value.

We know that for $X \in S$, the difference $\frac{\|N_{1:k}(X) - N^{app}_{1:k}(X)\|}{\|N_{1:k}(X)\|}$ is bounded by c. For any $X' \in I(X)$ and $X' \notin S$, $\frac{\|N_{1:k}(X') - N^{app}_{1:k}(X')\|}{\|N_{1:k}(X)\|}$ is not bounded and could be even more than c. Hence, even if $N_{1:k}(X), N^{app}_{1:k}(X'), N_{1:k}(X') \in T$, $N^{app}_{1:k}(X')$ can quite easily be outside the original template T. We intend to capture all such $N^{app}_{1:k}(X')$ inside our template for the approximate network by transforming the template T created on the original network.

We define the d-radius L_∞ ball around a point Y as $I_d(Y) = \{Y' | \|Y - Y'\|_\infty \leq d\}$. Suppose T is the template created using an input X in the template generation step. We join the box $I_d(N^{app}_{1:k}(X))$ around the point $N^{app}_{1:k}(X)$ with the original template T. We discuss the choice of d in Section 6. We use the standard join $\sqcup$ operator from $\mathcal{D}$ to combine two regions. The function α is the abstraction function for a domain $\mathcal{D}$. The new template value T^{app} can be represented by the following equation:

$$T^{app} = \gamma(\alpha(T) \sqcup \alpha(I_d(N^{app}_{1:k}(X))))$$

In FANC, we store templates as boxes (Section 4.3), and hence the abstraction function, the join operator and the concretization function have insignificant costs compared to the other steps. The transformation of the template does not alter the soundness of FANC. Irrespective of this transformation we always verify the validity of the templates before utilizing them for a particular approximate network (Algorithm 3). Our transformation heuristically modifies the templates such that a large fraction of these templates are expected to remain valid on the approximate network.

4.5 Verifying Approximate Network

Algorithm 3 presents FANC's procedure for verifying an approximate network. The algorithm takes a set of input regions ϕ as the input for verifier. It returns a boolean indicating if ψ holds for the region ϕ. The algorithm first filters out templates that are invalid for the approximate network. (line 1). The algorithm propagates each input region that needs to be verified to the layer k of the approximate neural network. If the abstract shape obtained by this propagation (concretized as $\mathcal{J}$) is contained in one of the valid templates then the proof is complete (lines 5-8).

If $\mathcal{J}$ is not contained in any template $T \in \mathcal{T}^{app}$, then we use the baseline analyzer on line 12. Computing $\mathcal{J}$ on line 4 and then running analyzer on $\mathcal{J}$ to the end on line 12 is the same as running the verification in the baseline from the input layer to the output layer. We next state the theorem that shows the soundness of the main step of the algorithm:

Theorem 2 (Soundness of Verification Algorithm 3). *If T is a valid template for neural network N^{app} and $\gamma(A_{\mathcal{D}}(I, N^{app}_{1:k})) \subseteq T$ then $\forall X \in I.\psi(N^{app}(X)) = true$.*

Proof.

$\gamma(A_{\mathcal{D}}(T, N^{app}_{k+1:l})) \vdash \psi$	(Since T is a valid template)
$\forall t \in T.N^{app}_{k+1:l}(t) \in \gamma(A_{\mathcal{D}}(T, N^{app}_{k+1:l}))$	(Using Definition 8)
$\forall t \in T.\psi(N^{app}_{k+1:l}(t)) = true$	(Combining the equations above)

Algorithm 3 Verifying approximate network

Input: N^{app}, property (ϕ,ψ), templates for approximate network $\mathcal{T}^{app}$, intermediate layer k
Output: Boolean indicating if the specification (ϕ,ψ) is verified

1: $\mathcal{T}^{app} = \{T \in \mathcal{T}^{app} \mid \gamma(A_{\mathcal{D}}(T, N^{app}_{k+1:l})) \vdash \psi\}$
2: **for** $I \in \phi$ **do**
3: b = false
4: $\mathcal{J} = \gamma(A_{\mathcal{D}}(I, N^{app}_{1:k}))$
5: **for** $T \in \mathcal{T}^{app}$ **do**
6: **if** $\mathcal{J} \subseteq T$ **then**
7: b = true
8: **break**
9: **end if**
10: **end for**
11: **if** $\neg b$ **then**
12: $b = \gamma(A_{\mathcal{D}}(\mathcal{J}, N^{app}_{k+1:l})) \vdash \psi$
13: **end if**
14: **if** $\neg b$ **then**
15: **return** false
16: **end if**
17: **end for**
18: **return** true

$\gamma(A_{\mathcal{D}}(I, N^{app}_{1:k})) \subseteq T$	(From the Theorem statement)
$\forall t \in \gamma(A_{\mathcal{D}}(I, N^{app}_{1:k})).\psi(N^{app}_{k+1:l}(t)) = true$	(Combining the equations above)
$\forall X \in I.N^{app}_{1:k}(X) \in \gamma(A_{\mathcal{D}}(I, N^{app}_{1:k}))$	(Using Definition 8)
$\forall X \in I.\psi(N^{app}_{k+1:l}(N^{app}_{1:k}(X))) = true$	(combining the last two equations)
$\forall X \in I.\psi(N^{app}(X)) = true$	

□

We further state the theorem that characterizes the precision of our algorithm:

Theorem 3 (Precision of Verification Algorithm 3). *Algorithm 3 is at least as precise as the baseline verifier V.*

Proof. This is an immediate consequence of the statement in line 12: whenever we cannot verify the input region using the template containment, we instead use the analyzer from the baseline verifier to verify it. Hence, our algorithm is at least as precise as the baseline verifier V. □

Although having more templates can improve the chances of the intermediate shape containment and will result in faster proof, the addition of each template (by the algorithms in Sections 4.3 and 4.4) also costs additional time for proving the template validity and checking the containment. Hence, it is important to balance this trade-off and find the right number of templates.

4.6 Time Complexity of FANC

The time complexity of FANC depends on the analyzer used in the verifier. In general, the complexity of the verifier for a given network and specification is a complicated function of the parameters of the network architecture (e.g., number of layers l and maximum number of neurons across all layers n) and specification (e.g., perturbation bounds, number of neurons that can take both negative and positive values). We make a simplifying assumption that for all networks with the same architecture, the verifier complexity for propagating abstract shapes through each layer is the same for all specifications and it depends on the maximum number of neurons across all layers. In this way, the cost of the verifier on a given network can be characterized using the network parameters l and n. Let $\tau_{A_\mathcal{D}}$ be a function that provides the analyzer cost as a function of n. Therefore, we assume that on average the analyzer takes $O(\tau_{A_\mathcal{D}}(n))$ time for propagation through one layer. A verifier using such an analyzer on a verification task on a neural network with l layers takes $O(l \cdot \tau_{A_\mathcal{D}}(n))$ time for running the analyzer. Checking if the output abstract shape can satisfy the property ψ can take different time based on the complexity of ψ. For this discussion, we assume that ψ is a robustness classification property (see Definition 4) – usually checking this property takes asymptotically lower time than running the analyzer [Singh et al. 2018b, 2019b], therefore we ignore it. Overall, a baseline verifier for an end-to-end verification task ϕ will take $O(l \cdot |\phi| \cdot \tau_{A_\mathcal{D}}(n))$ time. We divide this expression by $|\phi|$, and define the average time per interval specification $I(X) \in \phi$ for the baseline as

$$AVT_{bl} = O(l \cdot \tau_{A_\mathcal{D}}(n)).$$

We define λ as the containment fraction which is the fraction of the input regions that get captured in one of the templates in $\mathcal{T}^{app}$. Assuming that all the templates are created at layer k, the time complexity of running our fast verifier FANC is $O(\lambda \cdot k \cdot |\phi| \cdot \tau_{A_\mathcal{D}}(n) + (1-\lambda) \cdot l \cdot |\phi| \cdot \tau_{A_\mathcal{D}}(n))$. We do not add the containment checking time since it is asymptotically lower for our box shaped templates.

If ω_T is the number of iterations our template generator takes to create a single valid template for the original network, then its cost is $O(\omega_T \cdot l \cdot |\mathcal{T}| \cdot \tau_{A_\mathcal{D}}(n))$ time - since it performs validity check on each iteration. We evaluate the template generation time in Section 6. In the evaluation, we observe that ω_T is typically between 5 to 10.

The template transformer takes $O(n \cdot |\phi| \cdot k \cdot |\mathcal{T}|)$ time as the verifier needs to check the validity of transformed templates $\mathcal{T}^{app}$. This validity check takes $O(l \cdot |\mathcal{T}| \cdot \tau_{A_\mathcal{D}}(n))$ time. If we sum the time taken by all the components we get the overall complexity:

$$O(\omega_T \cdot l \cdot |\mathcal{T}| \cdot \tau_{A_\mathcal{D}}(n)) + O(n \cdot |\phi| \cdot |\mathcal{T}| \cdot k) + O(l \cdot |\mathcal{T}| \cdot \tau_{A_\mathcal{D}}(n)) + O(\lambda \cdot k \cdot |\phi| \cdot \tau_{A_\mathcal{D}}(n) + (1-\lambda) \cdot l \cdot |\phi| \cdot \tau_{A_\mathcal{D}}(n))$$

The analyzer time $\tau_{A_\mathcal{D}}(n)$ is usually at least quadratic [Singh et al. 2018b, 2019b; Wang et al. 2018b]. Hence, the second term for template transformation is asymptotically lower and can be ignored. The first term for template generation on the original network dominates the third term which represents the time for validating the transformed templates on the approximate network, and hence, we can ignore the third term asymptotically. Therefore the average time taken for verification per input specification $I(X) \in \phi$ is obtained by dividing the remaining terms with $|\phi|$. We get:

$$AVT_{pt} = O\left(\omega_T \cdot l \cdot \frac{|\mathcal{T}|}{|\phi|} \cdot \tau_{A_\mathcal{D}}(n)\right) + O\left(\lambda \cdot k \cdot \tau_{A_\mathcal{D}}(n) + (1-\lambda) \cdot l \cdot \tau_{A_\mathcal{D}}(n)\right).$$

If we achieve λ close to 1, then we get the minimum possible value for the second term in the expression for AVT_{pt}. If we compare AVT_{bl} and AVT_{pt}, we see that lower values of k, ω_T and $\frac{|\mathcal{T}|}{|\phi|}$ are essential for getting the maximum speedup.

Table 1. Models used for the evaluation.

Model	Architecture	Dataset	#Neurons
FCN7-MNIST	7×200 linear layers	MNIST	1400
FCN7-CIFAR	7×200 linear layers	CIFAR10	1400
CONV2-MNIST	2 Conv layers, 4 linear layers	MNIST	2456
CONV4-CIFAR	4 Conv layers, 4 linear layers	CIFAR10	8960

FANC works well for verification against low-dimensional perturbations that result in smaller templates to specifications ratio $\frac{|\mathcal{T}|}{|\phi|}$ (such perturbations includes patches, L_0, rotation and brightness). For L_∞ adversarial attacks, our online template generation strategy does not achieve a low enough value of $\frac{|\mathcal{T}|}{|\phi|}$ to achieve overall speedup.

5 METHODOLOGY

Networks: Table 1 presents the models used in evaluation. All the models are robustly trained using the training procedure from Chiang et al. [2020]. The approximate network versions are generated using post-training quantization and pruning. In the case of CNN, only the weight parameters corresponding to the fully-connected layers are pruned. For non-robustly trained networks, even the verification on the original network fails most of the time (as we empirically observed) and approximating such networks is unlikely to lead to robust networks. Since the verification is unlikely to succeed, we did not use such networks in our evaluation.

Perturbations: We used the patch, two versions of L_0 attack (L_0-random and L_0-center), rotation and brightness attacks presented in Section 3[2]:

- For the L_0-random attack, we first choose 3 as the threshold for perturbation. Proving robustness against all possible 3-pixel perturbation is currently impractical since it takes $\binom{729}{3}$ verification instance for one MNIST image. Hence, we perform verification against 1000 randomly sampled combinations of 3 pixels.
- For the L_0-center choose all the combinations of 3 pixels from the center of the picture since the center of the image contains an important part of the image in both MNIST and CIFAR10. We selected all $\binom{20}{3}$ set of 3 pixels from the central 5×4 patch in the image.
- For the patch attack, we verify the model against all perturbations in 2×2 patches.
- We consider rotation attack that rotates MNIST images at an angle $\gamma \in [-5, 5]$ degrees. Similar to Sprecher et al. [2021], we split these images to obtain precise certification. Currently, it is not possible to obtain CIFAR10 networks that are robust to rotation. Thus, we consider only MNIST images for the rotation experiment.
- The brightness attack is defined only on gray-scale MNIST images [Pei et al. 2017] . To increase certification precision we also use splitting of the brightness parameters.

Quantization: In this paper, we consider int8, int16 and float16 post-training quantizations. The quantization scheme is of the form [TFLite 2017]:

$$r = s(q - zp) \tag{4}$$

Here, q is the quantized value and r is the real value. s which is the scale and zp which is the zero point are the parameters of quantization. For our experiments, we use symmetric quantization that uses $zp = 0$. The quantization scheme uses a single set of quantization parameters for all values within each layer. At inference, weights are converted from quantized value to the floating point and

[2]The definitions from Section 3 assume a single color channel for notational convenience but they can be straightforwardly extended for images with multiple color channels like CIFAR10.

computations are performed using floating-point kernels. We randomly selected 25 images from the dataset for the quantization experiments. Each image generates 729 verification instances in case of a patch attack on MNIST, and similarly, there are multiple verification instances generated from a single image for other perturbations (L_0-random, L_0-center, rotation, brightness) as described earlier.
Pruning: At each iteration, we prune the smallest 10% model weights in each layer. Similar to many other iterative pruning techniques, we prune only affine layers (we do not prune weights in the convolution layer or biases, since they are much fewer in number than affine layer weights). We verify the robustness of each model against patch perturbation. We randomly select 10 images from the dataset and create input regions for every possible 2×2 patch perturbation. The number of input regions per image is 729 for MNIST and 961 for CIFAR10. In this experiment, we perform 10 pruning iterations. However, our technique can be used in any setting with many more iterations, if the transformed templates are valid and have a high value for the containment ratio λ.
Implementation: Code for our tool is written in Python and it is implemented on top of the ELINA library [Singh et al. 2018b, 2019b] for numerical abstractions. For all our experiments, we use DeepZ [Singh et al. 2018b] as the library underlying our analyzer and verifier. We use the zonotope abstract domain for the analysis. However, we store the templates using box abstraction since checking containment in the box is much faster. We do not expect to lose much precision by using box abstraction for storing templates, as we expect a high containment ratio λ in our experiments (and have observed it in Table 6).
Baseline: As the baseline, we use the plain DeepZ verifier for verification without proof transfer. We use the publicly available implementation of DeepZ in ERAN library [Singh et al. 2018a].
Hyperparameters: For getting the best possible performance we need to set the hyperparameter values. The possible values for the layer at which templates are created, k, is bounded by the number of layers in the model. Since we fix these values for each model and the search space is reasonably small, we choose them offline during the training.

Table 2. Hyperparameter values used by FANC

Model	κ	k
FCN7-MNIST	7	2
CONV2-MNIST	14	3
FCN7-CIFAR	16	3
CONV4-CIFAR	32	3

Table 2 presents the values of κ and k we used in the evaluation of each benchmark. For each network, we chose the hyperparameters by running FANC to verify L_0-random perturbations on a small subset of the images on the original network. We use the same hyperparameters for each network irrespective of the approximation and the attack. As a result, the hyperparameter tuning induces only a small overhead. For the tuning, we generate the templates using FANC and verify the original network. We selected κ by sweeping across fractions of the image width and selecting the best. We selected k by trying across values up to the number of layers and selecting the best.
Experimental Setup: For all the experiments we use 24 cores of an Intel Xeon E5-2687W CPU with a main memory of 64 GB running Linux operating system.

6 EXPERIMENTAL EVALUATION

We evaluates the effectiveness of FANC on verifying quantized and pruned networks. We then analyze how various tool components contribute to the overall result.

6.1 Effectiveness of FANC

Analysis and Verification Times: Tables 3 and 4 summarize the verification results for different quantization approximations. In each table, Column T_{Base} is the time for running the baseline verification. Column T_{Fanc} is the time for verification using proof transfer over the network. Column Sp represents the speedup obtained by FANC's core verification with proof transfer compared to the

Table 3. Average time (seconds) for verification for L_0-random, L_0-center and patch attacks.

Model	Approximation	L_0-random			L_0-center			patch		
		T_{Base}	T_{Fanc}	Sp	T_{Base}	T_{Fanc}	Sp	T_{Base}	T_{Fanc}	Sp
FCN7-MNIST	float16	41.44	13.67	3.03	47.84	18.16	2.63	30.71	13.66	2.25
	int16	42.58	12.99	3.28	47.76	21.46	2.23	30.78	13.74	2.24
	int8	40.99	10.44	3.93	47.31	30.22	1.57	30.28	11.5	2.63
CONV2-MNIST	float16	47.81	27.78	1.72	54.77	35.71	1.53	35.07	24.85	1.41
	int16	47.78	27.71	1.72	54.89	36.04	1.52	34.74	24.74	1.4
	int8	47.61	29.63	1.61	54.25	37.97	1.43	34.29	25.56	1.34
FCN7-CIFAR	float16	72.3	48.12	1.5	72.33	48.05	1.51	80.66	49.32	1.64
	int16	69.52	39.73	1.75	70.97	40.53	1.75	71.25	48.40	1.47
	int8	70.31	70.31	1	79.69	69.88	1.14	78.28	69.21	1.13
CONV4-CIFAR	float16	142.91	104.43	1.37	160.38	125.02	1.28	150.43	106.53	1.41
	int16	148.21	106.51	1.39	160.77	128.3	1.25	151.21	108.72	1.39
	int8	138.81	116.57	1.19	163.26	129.42	1.26	149.84	117.13	1.28

Table 4. Average time (seconds) for verification for rotation and brightness attacks.

Model	Approximation	rotation			brightness		
		T_{Base}	T_{Fanc}	Sp	T_{Base}	T_{Fanc}	Sp
FCN7-MNIST	float16	46.54	19.43	2.40	38.09	9.22	4.13
	int16	42.80	21.10	2.03	36.81	11.33	3.25
	int8	32.92	18.66	1.76	33.33	11.89	2.8
CONV2-MNIST	float16	52.34	41.02	1.28	54.51	21.73	2.51
	int16	48.32	36.68	1.32	41.34	20.68	2.0
	int8	42.84	29.86	1.43	40.16	26.56	1.51

baseline (without the template creation). In each case, the time in the table is averaged over all the images used in the experiment. Tables 3 and 4 do not add the template generation time to the FANC verification time. We generate templates on the original network and leverage them to speed up its verification. We reuse the templates for the verification of all other approximate networks in Tables 3 and 4. We present the end-to-end savings with the template generation times included in Table 5.

In most quantized networks proof transfer results in better verification time. For example, in Table 3, for CONV2-MNIST network we get about 1.72x speedup for both float16 and int16 quantization on the L_0-random attack. Similarly, we see 1.45x speedup on the L_0-center attack.

We also perform experiments where we skip the template transformation step and use the generated templates without any modification. We observe that the transformation step contributes to a major improvement in the speedup. The total speedup with the template transformation step is 1.5x and without it is 1.26x.

End-to-end Speedup: To present a realistic use case and end-to-end time savings we consider the task of verifying the original and one approximate network. The baseline verifier needs to be executed twice. In contrast, with FANC, the verification consists of (1) running the verifier on the original network, which also creates the template, and (2) running the verifier on the approximate network using the template. The speedup is in this case:

$$Sp_{tot} = \frac{T_{Base}(N) + T_{Base}(N^{app})}{T_{Fanc}(N) + T_{Fanc}(N^{app}) + T_{tg}} \tag{5}$$

Table 5. The end-to-end speedup for each attack on quantization when verifying the original and a single approximate network.

Model	Approximation	End-to-end speedup (Sp_{tot}) L_0-random	L_0-center	patch	rotation	brightness
FCN7-MNIST	float16	2.17	1.95	1.8	1.87	2.27
	int16	2.23	1.82	1.8	1.74	2.11
	int8	2.35	1.56	1.91	1.63	1.98
CONV2-MNIST	float16	1.54	1.29	1.25	1.17	2.06
	int16	1.54	1.28	1.25	1.18	1.85
	int8	1.49	1.25	1.22	1.22	1.64
FCN7-CIFAR	float16	1.18	1.15	1.29	-	-
	int16	1.25	1.21	1.3	-	-
	int8	0.99	1.03	1.1	-	-
CONV4-CIFAR	float16	1.14	1.09	1.2	-	-
	int16	1.15	1.08	1.19	-	-
	int8	1.07	1.08	1.15	-	-

Table 6. The containment fraction λ for each attack on quantization.

Model	Approximation	$\hat{c}$	Containment ratio λ L_0-random	L_0-center	patch	rotation	brightness
FCN7-MNIST	float16	2.0e-7	0.99	0.99	0.99	0.95	0.99
	int16	2.0e-6	0.99	0.99	0.99	0.91	0.99
	int8	0.038	0.98	0.94	0.91	0.86	0.98
CONV2-MNIST	float16	2.4e-6	0.72	0.68	0.71	0.54	0.98
	int16	1.2e-5	0.72	0.67	0.70	0.41	0.97
	int8	0.116	0.68	0.62	0.68	0.52	0.97
FCN7-CIFAR	float16	2.3e-3	0.91	0.79	0.88	-	-
	int16	0.078	0.89	0.74	0.86	-	-
	int8	0.23	0.64	0.61	0.54	-	-
CONV4-CIFAR	float16	7.9e-5	0.69	0.63	0.59	-	-
	int16	7.4e-4	0.66	0.55	0.53	-	-
	int8	0.036	0.49	0.49	0.46	-	-

We measured the time for template creation and template transfer (which happen only once). The template creation time (T_{tg}) for FCN7-MNIST is 11.21s, for CONV2-MNIST 6.93s, for FCN7-CIFAR 26.48s and for CONV4-CIFAR 31.62s. These times are small relative to the verification time even for a single (original) network, resulting often (especially for the MNIST networks) in faster verification than with the baseline.

Table 5 presents the end-to-end savings for this scenario. For each approximate network, the columns present the speedups obtained by each transformation. The speedups for the networks are between 1.1x and 2.27x. The only exception is the FCN7-CIFAR network for int8 approximation, which is due to the high dissimilarity of the original and the approximate networks. The cost of template creating amortizes further when verifying multiple approximations, since FANC can reuse the template (the results from Tables 3 and 4 present the upper bound).

Impact of the Containment Fraction: Table 6 presents the value of constant λ for each attack on the quantization approximations. Columns 1 and 2 present the network and the approximation. Column 3 presents the approximation constant for the network (lower indicates more similar networks; Section 4.1). Columns 4-7 present the λ value for each attack (higher is better). The speedup

Table 7. Average time (seconds) for verification of 10 pruned MNIST models.

Model	FCN7-MNIST				CONV2-MNIST			
Prune(%)	$\hat{c}$	T_{Base}	T_{Fanc}	Sp	$\hat{c}$	T_{Base}	T_{Fanc}	Sp
0	0	31.71	11.31	2.8	0	34.97	23.36	1.5
10	3e-3	31.07	11.42	2.72	5e-4	34.73	23.38	1.49
20	1e-2	30.64	11.13	2.75	4e-3	34.58	23.09	1.5
30	2.7e-2	30.36	11.02	2.75	9e-3	34.68	22.6	1.53
40	4.7e-2	30.14	10.76	2.8	1.7e-2	34.32	23.16	1.48
50	6.9e-2	29.77	10.75	2.77	3.9e-2	34.48	22.8	1.51
60	0.1	29.49	10.62	2.78	8.6e-2	34.41	23.1	1.49
70	0.16	28.62	26.26	1.09	0.16	34.12	22.98	1.48
80	0.27	28.32	29.13	0.97	0.27	34.25	25.21	1.36
90	0.62	27.39	27.67	0.99	0.47	33.48	34.67	0.97
End-to-end		297.52	171.27	1.73x		344.02	251.28	1.36x

Table 8. Average time (seconds) for verification of 10 pruned CIFAR10 models.

Model	FCN7-CIFAR				CONV4-CIFAR			
Prune(%)	$\hat{c}$	T_{Base}	T_{Fanc}	Sp	$\hat{c}$	T_{Base}	T_{Fanc}	Sp
0	0	80.98	43.27	1.87	2.3e-4	147.96	105.05	1.41
10	8.1e-4	77.13	42.11	1.83	2.3e-4	149.18	106.75	1.4
20	2.8e-3	73.46	44.71	1.64	2.3e-4	149.88	105.56	1.42
30	4.2e-3	70.53	41.21	1.71	2.3e-4	149.48	105.85	1.41
40	1.1e-2	67.47	41.71	1.62	2.3e-4	148.19	105.68	1.4
50	1.5e-2	64.3	41.52	1.55	2.3e-4	150.75	104.9	1.44
60	1.6e-2	60.92	41.82	1.46	1.4e-2	149.16	105.88	1.41
70	4.1e-2	57.97	41.39	1.4	1.4e-2	148.49	105.34	1.41
80	7.8e-2	53.59	42.23	1.27	1.5e-2	147.83	105.47	1.4
90	0.3	48.95	42.11	1.16	1.7e-2	146.13	111.21	1.31
End-to-end		655.31	448.56	1.46x		1487.04	1093.3	1.36x

of the proof transfer is positively correlated with the containment fraction λ for a fixed model. For FCN7-MNIST we get λ close to 1 for all of the quantization schemes. One example where the proof transfer does not work well is FCN7-CIFAR on int8 quantization. In this case, the approximation constant $\hat{c}$ is 1.92, which indicates the network is quite dissimilar from the original network.

6.2 Multiple Network Proof Transfer for Pruning

In this section, we show that how proof transfer can be used in a setting that continuously changes the model by pruning. For these experiments, we use the patch attack.

Table 7 presents the average verification times for all the MNIST models. Table 8 presents the average verification times for all the CIFAR10 models. We use the same notation T_{Base}, T_{Fanc} and Sp from Section 6.2 for the columns. Row "End-to-end" for Column T_{Fanc} shows the total time for verification, including the template generation time, and for Column Sp shows the total end-to-end speedup that we get (as computed by Equation 5).

In Tables 7 and 8, we present the $\hat{c}$ value for each of the networks (Section 4.1). It shows that FANC speedup is inversely proportional to $\hat{c}$. For a particular network, one could easily compute $\hat{c}$ in negligible time as a heuristic before using FANC and estimate the likelihood of obtaining speedup with FANC. For instance, if we know that FANC on an approximate network version N_1 with $\hat{c} = 0.1$ obtains 2x speedup then another approximate network version N_2 with $\hat{c} < 0.1$ will likely obtain more than 2x speedup.

For instance, in the Table 7, when we prune 40% of model weights in the FCN7-MNIST network then $\hat{c} = 0.047$ and our technique gives about 3x speedup. However, when we prune 70% of model weights, then $\hat{c} = 0.1$ and our technique gives only 1.1x speedup. In most cases, the speedups are higher with the template transformation. In Table 8, if we prune FCN7-CIFAR network 30%, then the speedup of the verifier without the template transformation is about 1.4x, and the speedup with the transformation is about 1.7x. A similar pattern is observed in all the cases for CONV2-MNIST, FCN7-CIFAR and CONV4-CIFAR. The speedup is obtained even if we have to perform verification of a single approximate network. However, we emphasize that the same set of templates can work on a class of possible approximate networks.

6.3 Template Transfer

In this section, we evaluate the hypothesis that a significant fraction of the templates generated using the original network after transformation are valid for approximate networks. We expect that the ratio $\eta = \frac{|valid(\mathcal{T}^{app})|}{|\mathcal{T}^{app}|}$ is to be higher for float16 and int16 quantization, and it drops for int8 quantization. Table 9 presents the result for each quantization scheme and the network. Each cell contains the ratio η for each network and approximation. As expected, we observe that the template transfer ratio is higher for float16 and int16 quantization and reduces somewhat in the case of int8 quantization.

Table 9. The template transfer ratio $\eta = \frac{|valid(\mathcal{T}^{app})|}{|\mathcal{T}^{app}|}$ for all models and quantization combinations.

Model	float16	int16	int8
FCN7-MNIST	0.75	0.75	0.78
CONV2-MNIST	0.84	0.84	0.85
FCN7-CIFAR	0.54	0.54	0.28
CONV4-CIFAR	0.48	0.48	0.47

Table 10. Average verification speedup $Sp = \frac{T_{Fanc}}{T_{Fanc\ [sp]}}$ due to Algorithm 2 over template generation from Sprecher et al. [2021]

Attack	int8	int16	float16	prune-50%
L_0-random	1.27x	1.15x	1.11x	1.06x
L_0-center	1.17x	1.18x	1.15x	1.1x
patch	1.15x	1.06x	1.04x	1.07x
rotation	1.05x	1.07x	1.18x	1.12x
brightness	1.1x	1.1x	1.12x	1.15x

6.4 Ablation Study

In this section, we present an ablation study comparing instantiating FANC with Algorithm 2 for the template generation step and FANC with Sprecher et al. [2021] (which is a special case of Algorithm 2 when $\kappa = h = w$), keeping all other components the same. Sprecher et al. [2021] generates a single template per image. Whereas FANC can generate multiple templates per image. Our main motivation for using Algorithm 2 to generate multiple templates is that if some of the templates are not valid for the approximate network then other templates could be used instead. We perform the experiment on all combinations of attacks and all approximations. Tables 10 and 11 present the results of our ablation study.

Each entry in Table 10 presents the average speedup across all verification instances of a particular perturbation type (shown as rows) and all models with the same approximation (shown as columns).The speedup of each instance is computed as $Sp = \frac{T_{Fanc}}{T_{Fanc\ [sp]}}$ where $T_{Fanc}, T_{Fanc\ [sp]}$ are the time taken by FANC verification with template generation from Algorithm 2 and Sprecher et al. [2021] respectively. Overall, the results show that Algorithm 2 yields improvement across all approximations and perturbations, ranging between 1.27x (int8 with L_0-random attack) and 1.04x (float16 with patch attack). Sprecher et al. [2021] template generation algorithm results in smaller containment ratios for more aggressive approximations (higher value of $\hat{c}$). For these cases with

Table 11. Average verification speedup due to Algorithm 2 over Sprecher et al. [2021] template generation for int8 quantization

Attack	Model	Sp	$\lambda_{FANC[sp]}$	λ_{FANC}
L_0-random	FCN7-MNIST	1.64x	0.73	0.98
	CONV2-MNIST	1.13x	0.61	0.68
	FCN7-CIFAR	1.3x	0.52	0.67
	CONV4-CIFAR	1x	0.49	0.49
L_0-center	FCN7-MNIST	1.31x	0.84	0.94
	CONV2-MNIST	1.16x	0.46	0.62
	FCN7-CIFAR	1.18x	0.61	0.64
	CONV4-CIFAR	1x	0.49	0.49
patch	FCN7-MNIST	1.35x	0.74	0.91
	CONV2-MNIST	1.14x	0.55	0.68
	FCN7-CIFAR	1.1x	0.47	0.61
	CONV4-CIFAR	1x	0.46	0.46
rotation	FCN7-MNIST	1.1x	0.69	0.86
	CONV2-MNIST	1x	0.43	0.52
brightness	FCN7-MNIST	1x	0.99	0.99
	CONV2-MNIST	1.2x	0.8	0.99

larger values of $\hat{c}$, Algorithm 2 shows more improvement in the containment ratios and results in higher speedup. In these cases having more templates is useful in improving the containment ratio. As indicated by the $\hat{c}$ values from Table 6 and Table 7, prune-50% network is less approximate than the quantized network for each model, and therefore results in a smaller improvement with Algorithm 2.

Since the results in Table 10 show that our Algorithm 2 is particularly useful in the case of int8 quantization, we present detailed results for the verification of each network in Table 11. Column $\lambda_{FANC[sp]}$ presents the template containment ratio for the templates generated using the algorithm from Sprecher et al. [2021]. Column λ_{FANC} presents the ratio for the templates generated using the Algorithm 2. The results show a substantial improvement in the containment ratio λ due to Algorithm 2 that results in the consistent speedup across different networks. In particular, the Pearson correlation coefficient between speedup and the differences of λ is 0.8 (highly positive on the scale from -1.0 to 1.0). In Table 11, CONV4-CIFAR cases do not yield speedups since the best choice of $\kappa = 32$ is the same as in Sprecher et al. [2021]. For these cases, the containment ratio λ is higher for $\kappa = 32$ (average $\lambda = 0.49$) than for $\kappa = 16$ (average $\lambda = 0.44$) or $\kappa = 8$ (average $\lambda = 0.45$). Hence, generating multiple templates by using smaller κ does not always improve the containment ratio. Finally, for FCN7-MNIST under the brightness perturbation, the verification problem is simple, and even a coarse κ leads to a high containment ratio ($\lambda = 0.99$).

7 RELATED WORK

Proof Transfer: Sprecher et al. [2021] introduced the concept of proof transfer for neural network certification between different input specifications. This approach enables proof effort reuse across input specifications to reduce the overall verification cost between 1.2x and 2.9x on neural networks with fully-connected layers. However, this approach cannot transfer proofs across different networks. Further, it is primarily aimed at networks with fully-connected layers, with a limited support for convolutional layers – the proof transfer can occur only after all the convolutional layers, which significantly reduces its applicability. FANC uses the idea of proof transfer to multiple approximate networks (as opposed to different inputs on the single network). The output values and the intermediate activation values of the networks change due to the approximation. Thus, for the verification of the approximate network, we cannot use templates created on the original network.

To perform a sound transfer of the proof, we need templates that are valid on the new network. Our unique template transformation approach enables modification of proof templates to work for a specific approximate network. Hence, transforming the templates is essential in transferring the proof to approximate networks. Our template generation algorithm can be applied for networks with convolutional layers. The template generation algorithm from Sprecher et al. [2021] can be encoded as a special case of our algorithm when $\kappa = h = w$.

Ashok et al. [2020] presented an approach transferring proof from an abstraction of a network to the original network. The framework creates abstractions by clustering from experimental data; the abstractions relate the neurons in (simpler) abstract network to the neurons in the original network. As one application of their work, they show how one can transfer a proof from the abstract network to the original one. The evaluation demonstrated that the approach works for fully-connected feedforward neural network with 1.25x speedup. The goal of FANC is different: it transfers the proof from the original network to an approximate network by creating an abstraction that aims to over-approximate the network behavior. Moreover, our technique works on much larger models with convolutions, and in the case of fully-connected networks we show up to 4.1x speedup.

Cheng and Yan [2020] reuse previous analysis results after the weights of neural network with fully-connected layers are fine-tuned (retrained) in a continuous setting (the paper presents a case study of a visual pipeline in autonomous vehicle model). In comparison to our work, their verification goal is different and the models that the technique supports are significantly limited: their verifier is used only for the verification of fully-connected linear layers at the end of the network. In contrast, FANC operates on a more general class of networks and approximations, and end-to-end verifies networks with both fully-connected and convolutional layers.

Differential neural network verification is a related concept that aims to bound the difference between the output of original and approximate networks [Paulsen et al. 2020a,b]. However, this approach and the solving mechanisms cannot be directly used to prove the robustness properties of the approximate network.

Neural Network Verification: Researchers have proposed several techniques for verifying properties of neural networks [Anderson et al. 2019, 2020; Bunel et al. 2020; Ehlers 2017; Katz et al. 2017; Laurel et al. 2022; Singh et al. 2019c; Tjeng et al. 2017; Wang et al. 2018a, 2021]. To overcome the inherent scalability limitations and support a broader range of neural networks, state-of-the-art verifiers have explored the trade-offs between scalability and precision [Singh et al. 2019a,b; Tjandraatmadja et al. 2020; Xu et al. 2020; Zhang et al. 2018], or computing estimates of robustness [Baluta et al. 2021; Bastani et al. 2016; Weng et al. 2018b,a]. We anticipate that the improvements in the verifier technology will lead to more interest in analyzing both the original and approximate networks. While our implementation used a particular DeepZ verifier, most state-of-the-art incomplete verifiers (e.g., Fast-Lin [Weng et al. 2018a], CROWN [Zhang et al. 2018], Neurify [Wang et al. 2018a], Star sets [Tran et al. 2020]) can be reformulated as instances of abstract interpretation. The idea of template generation and template transfer are general and can in principle be used with other verifiers to save the overall verification effort and time.

8 CONCLUSION

The requirement for running neural networks on energy-constrained devices, or shifts in the input distribution, will continue to drive the development of new approaches for approximating and tuning neural networks. The current approaches for verifying neural networks, which simply re-run the proofs from scratch, will not be able to keep up with the rate at which the networks are modified during deployment.

In this paper, we presented FANC, the first general technique for transferring proofs between the original neural network and its multiple approximated versions. FANC's algorithms create templates

(connected symbolic shapes at an intermediate network layers that capture the proof of the property to verify) and present algorithms for efficient template transformations and checking. We evaluated the effectiveness of our approach for verifying networks generated by various approximation techniques, including quantization and pruning, on fully-connected and convolutional networks, and against different adversarial attacks, including two L_0 attacks, patch, rotation and brightness attacks. Our results indicate that FANC can significantly improves verification speed, up to 4.1x with median 1.55x for quantization and up to 2.8x with median 1.48x for pruning.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments. This research was supported in part by NSF Grants No. CCF1846354, CCF-1956374, CCF-2008883, USDA NIFA Grant No. NIFA-2024827 and a gift from Facebook.

REFERENCES

Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. 2013. Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine* 11, 2 (2013), 47–58.

Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. 2019. Optimization and Abstraction: A Synergistic Approach for Analyzing Neural Network Robustness. In *Proc. Programming Language Design and Implementation (PLDI)*. 731–744.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. 2020. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming* (2020), 1–37.

Pranav Ashok, Vahid Hashemi, Jan Kretínský, and Stefanie Mohr. 2020. DeepAbstract: Neural Network Abstraction for Accelerating Verification. In *Automated Technology for Verification and Analysis - 18th International Symposium*, Vol. 12302.

Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. 2019. Certifying Geometric Robustness of Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 32.

Teodora Baluta, Zheng Leong Chua, Kuldeep S Meel, and Prateek Saxena. 2021. Scalable quantitative verification for deep neural networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 312–323.

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. *Advances in neural information processing systems* 29 (2016), 2613–2621.

Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. 2020. What is the State of Neural Network Pruning?. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020*.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Pushmeet Kohli, P Torr, and P Mudigonda. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research* 21 (2020).

Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *IEEE S&P Symposium*. 39–57.

Chih-Hong Cheng and Rongjie Yan. 2020. Continuous Safety Verification of Neural Networks. arXiv:2010.05689 [cs.LG]

Ping-yeh Chiang, Renkun Ni, Ahmed Abdelkader, Chen Zhu, Christoph Studor, and Tom Goldstein. 2020. Certified Defenses for Adversarial Patches. In *International Conference on Learning Representations*.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified Adversarial Robustness via Randomized Smoothing. In *Proceedings of the 36th International Conference on Machine Learning*.

Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. 269–286.

Mikhail Figurnov, Aizhan Ibraimova, Dmitry P. Vetrov, and Pushmeet Kohli. 2016. PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions. In *Advances in Neural Information Processing Systems 2016*. 947–955.

Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *Proc. International Conference on Learning Representations (ICLR)*.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *IEEE S&P Symposium*. 3–18.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. *CoRR* abs/2103.13630 (2021).

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations, ICLR 2015*.

ISO 2021. *Assessment of the robustness of neural networks*. Standard. International Organization for Standardization.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2018. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *CoRR* abs/1810.04240 (2018).

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification - 29th International Conference, CAV*, Vol. 10426. 97–117.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *International Conference on Computer Aided Verification, CAV*, Vol. 11561. 443–452.

Jacob Laurel, Rem Yang, Gagandeep Singh, and Sasa Misailovic. 2022. A dual number abstraction for static analysis of Clarke Jacobians. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–30.

Mathias Lécuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified Robustness to Adversarial Examples with Differential Privacy. In *IEEE S&P Symposium*. 656–672.

Jingyue Lu and M. Pawan Kumar. 2020. Neural Network Branching for Neural Network Verification. In *International Conference on Learning Representations*.

Antoine Miné. 2001. The Octagon Abstract Domain. In *Working Conference on Reverse Engineering, WCRE'01*. 310.

Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020a. ReluDiff: differential verification of deep neural networks. In *ICSE '20: 42nd International Conference on Software Engineering*.

Brandon Paulsen, Jingbo Wang, Jiawei Wang, and Chao Wang. 2020b. NEURODIFF: Scalable Differential Verificatfion of Neural Networks using Fine-Grained Approximation. In *International Conference on Automated Software Engineering*.

Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. 2019. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In *Proc. Neural Information Processing Systems (NeurIPS)*. 1394–1406.

Hadi Salman, Jerry Li, Ilya P. Razenshteyn, Pengchuan Zhang, Huan Zhang, Sébastien Bubeck, and Greg Yang. 2019a. Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers. In *Proc. Neural Information Processing Systems (NeurIPS)*. 11289–11300.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019b. A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In *Advances in Neural Information Processing Systems 32*. 9832–9842.

Hashim Sharif, Yifan Zhao, Maria Kotsifakou, Akash Kothari, Ben Schreiber, Elizabeth Wang, Yasmin Sarita, Nathan Zhao, Keyur Joshi, Vikram S. Adve, Sasa Misailovic, and Sarita V. Adve. 2021. ApproxTuner: a compiler and runtime system for adaptive approximations. In *PPoPP '21: 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 262–277.

Gagandeep Singh et al. 2018a. ERAN. https://github.com/eth-sri/eran.

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. 2019a. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems*. 15098–15109.

Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018b. Fast and Effective Robustness Certification. In *Advances in Neural Information Processing Systems*, Vol. 31.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019b. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.* 3, POPL (2019).

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019c. Boosting Robustness Certification of Neural Networks. In *International Conference on Learning Representations*.

Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2017. Fast polyhedra abstract domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL*. 46–59.

Christian Sprecher, Marc Fischer, Dimitar I. Dimitrov, Gagandeep Singh, and Martin Vechev. 2021. Proof Transfer for Neural Network Verification. arXiv:2109.00542 [cs.LG]

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations*.

TFLite. 2017. TF Lite post-training quantization. https://www.tensorflow.org/lite/performance/post_training_quantization.

Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Patel, and Juan Pablo Vielma. 2020. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *arXiv preprint*

arXiv:2006.14076 (2020).

Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2017. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356* (2017).

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations, ICLR 2019*.

Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *Proc. Computer Aided Verification (CAV)*. 18–42.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018a. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*. 6367–6377.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018b. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *27th USENIX Security Symposium, USENIX Security*. 1599–1614.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. *arXiv preprint arXiv:2103.06624* (2021).

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. 2018b. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. In *Proc. International Conference on Learning Representations, ICLR 2018*.

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. 2018a. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699* (2018).

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*. 4944–4953.