International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

# Benchmarking provable resilience in convolutional neural networks: A study with Beta-CROWN and ERAN

Santosh Appachu Devanira Poovaiah *

*University of Southern California, Los Angeles, California, USA.*

International Journal of Science and Research Archive, 2022, 07(02), 834-845

Publication history: Received on 11 September 2022; revised on 20 November 2022; accepted on 26 November 2022

Article DOI: https://doi.org/10.30574/ijsra.2022.7.2.0319

## Abstract

Robust verification of neural networks is a critical property, particularly in the context of safety-critical applications. Despite its importance, achieving both accurate and efficient verification remains a significant challenge. This study focuses on the robustness of convolutional neural networks (CNNs), which are widely deployed in domains such as autonomous driving and medical diagnostics, where failure is not an option. We explored two state-of-the-art verification techniques, namely Beta-CROWN and ERAN, that leverage linear approximation methods to handle the inherent non-linearity of neural networks. Both approaches approximate nonlinear constraints to enable tractable formal verification. A comparative analysis is conducted to evaluate their performance in terms of certification accuracy and computational efficiency. Our findings provide insights into the practical trade-offs between these techniques and guide future work in provable resilience of CNNs.

**Keywords:** Convolutional neural networks; Certified robustness; Formal verification; Perturbations; Rectified Linear Unit (ReLU); Beta-CROWN; ERAN

## 1. Introduction

Numerous applications, such as pattern recognition, control systems, picture classification, speech processing, and audio analysis, now depend heavily on neural networks. Their capacity to learn hierarchical representations and approximate complex functions has led to their growing use in safety-critical fields like industrial automation, medical diagnosis, and autonomous driving. Convolutional neural networks (CNNs) have shown impressive performance in tasks involving visual perception, allowing systems to make decisions that are comparable to those made by humans. Nevertheless, despite their remarkable capabilities, CNNs have a well-established susceptibility to adversarial perturbations, which are minute, expertly designed changes to input data that are frequently undetectable to humans but can cause the network to misclassify data incorrectly or even dangerously. Particularly in mission-critical settings where forecast errors might have dire repercussions, this lack of resilience raises serious safety and reliability issues. To address this, formal robustness verification methods have emerged as a vital research direction. These techniques aim to mathematically prove that, for all inputs within a certain bounded region, the network's output remains consistent or satisfies specified safety properties. Robust verification thus ensures that CNNs behave reliably under bounded input perturbations and helps build trust in their deployment in real-world scenarios.

Several techniques have been proposed for verifying convolutional neural networks. This paper discusses two such methods: ERAN and Beta-CROWN. It presents a comparative analysis of these methods, focusing on their algorithmic implementation as well as their performance in terms of computational efficiency and accuracy.

* Corresponding author: Santosh Appachu Devanira Poovaiah

## 2. Related Work

In contrast to Beta-Crown, previous work on neural network verification often relied on Mixed Integer Linear Programming (MILP) solvers [2] [3] [9]. But their applicability is restricted to very small networks. Researchers have also suggested methods based on branch and bound in the past [5] [11]. But for these methods, they have been limited by input dimensions. There have been papers where the verification problem has been decomposed layer by layer [12] and each layer has been solved in a closed form on GPUs and used Lagrangian to enforce consistency between layers. Beta-Crown's algorithms were developed to address the limitations of these prior methods, which, despite their formulation having only linear programming power, required many iterations to converge and incur higher computational times. When comparing previous works with ERAN, there have been many studies on network robustness based on an abstraction-refinement strategy for feedforward neural networks (FNN) [7]. However, this was demonstrated to be successful for a network of only six neurons. [14] extended the simplex algorithm to test FNN properties using ReLU. [1] demonstrated a verification framework based on an SMT solver that validated robustness with respect to a specific set of functions that can manipulate the input and are minimal. However, it is unclear how such a set can be obtained. Many works deal with robust analysis of programs. [15] considered a definition of robustness like the one used in ERAN , and [13] demonstrated program robustness using a combination of abstract interpretation and SMT-based methods. It is challenging to directly compare using a similar method to neural networks because the programs studied in this area are usually small. In that they lack sophisticated language properties but have the capacity to grow enormously, neural networks are more akin to circuits.

## 3. Approach and Algorithm

This paper undertook a comparative analysis of two prominent neural network verifiers: Beta-CROWN and ERAN. The study aimed to thoroughly evaluate these tools to understand their respective strengths and weaknesses when applied to complex verification tasks. By examining their underlying algorithms, computational efficiency, and accuracy across various network architectures and robustness properties, we sought to provide a comprehensive assessment of their performance. This detailed comparison offers valuable insights for researchers and practitioners looking to choose the most suitable verification method for their specific needs.

### 3.1. Beta-CROWN Analysis

A notable development in neural network verification, beta-CROWN is mostly known for its excellent scalability and efficiency in delivering verifiable robustness assurances. It uses highly optimized linear bound propagation to compute rigorous lower and upper bounds for network outputs within designated input perturbation regions, building upon and expanding the CROWN (Complete/Conditional ReLU Optimization with Neuron-split) and LiRPA (Linear Relaxation based Perturbation Analysis) frameworks. A key innovation of Beta-CROWN lies in its ability to directly encode neuron split constraints from a Branch-and-Bound procedure into its bound propagation using optimizable β parameters. This design elegantly replaces the computationally expensive Linear Programming (LP) solves traditionally required for handling these splits, allowing it to achieve bounds comparable to or tighter than LP-based methods but with vastly superior efficiency and parallelizability, particularly on GPUs. When used with Branch-and-Bound search, Beta-CROWN offers comprehensive verification, able to find counterexamples or conclusively prove characteristics. It can scale to much larger convolutional networks than was previously possible with traditional Mixed Integer Linear Programming (MILP) or LP-based verifiers, overcoming their limitations regarding computational time and applicability to only very small networks. This is made possible by the potent combination of fast, optimizable bound propagation and a robust search strategy. The basic idea behind Beta-Crown for neural network verification is to reduce the verification problem to find the global minima of some functions on the network's output, within a bounded set of constraints. Let us assume that we are given: $f\ (x0)\ >\ 0$, we need to verify that,

$$f(x) > 0, \quad \forall x \in C \qquad \text{..... (1)}$$

Here $C$ is the perturbation set around $x0$. To verify equation (1) we need to solve the optimization problem to find the worst case:

$$f^* = \min_{x \in C} f(x) \qquad \text{..... (2)}$$

If we can demonstrate that the value of equation (2) is positive, then the network can be confidently classified as provably robust. This means the network reliably meets the required conditions under all specified scenarios. On the other hand, if this positive value cannot be established, the network is considered non-robust, indicating potential

vulnerabilities or failures under certain conditions. Establishing this clear distinction helps us understand the reliability of the network and guides future improvements to enhance its stability and performance.

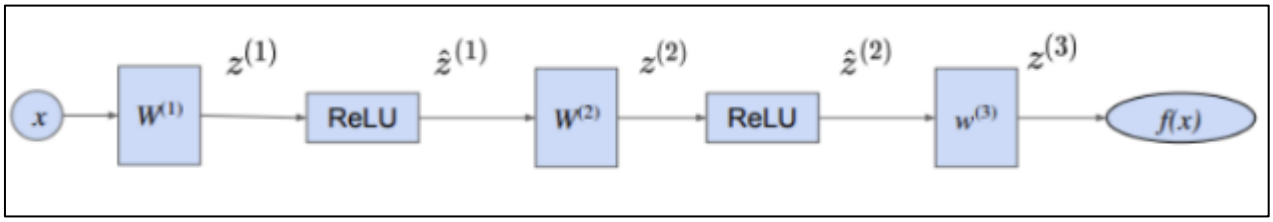



**Figure 1** Simple neural network

Consider the neural network outlined in Fig.1. Due to the presence of the ReLU activation function in the network, the problem introduces non-linear and non-convex constraints. These characteristics significantly increase the complexity of solving equations (2) because traditional optimization techniques often rely on linearity or convexity to guarantee efficient and reliable solutions. The non-convex nature means there can be multiple local minima, making it difficult to find the global minimum with certainty. Some methods to deal with this use solvers based on Mixed Integer Programming (MIP), which can precisely represent the ReLU constraints and offer solutions. Despite their strength, MIP-based solvers have scaling problems and a high computational cost. The time and resources needed to solve the problem using MIP might become unaffordable for large-scale networks or complicated models, which restricts their practical use. More effective algorithms or approximations that strike a balance between accuracy and computing viability are thus required. Therefore, we use the method of approximation to solve the minimization problem. The approach can be divided into two sub-parts where we first define the working of Crown, and then we add some additional constraints to incorporate the factor Beta to completely define the method as Beta-Crown.

*3.1.1. Crown*

Carrying forward the approach of approximation, Crown [4] aims to find a certified lower bound such that:

$$f^*_{\text{CROWN}} \leq f^* = \min_{x \in C} f(x) \qquad (3)$$

We have to make sure that $f^*_{\text{CROWN}}$ is positive under all circumstances. The output would basically be a straightforward composite of linear transformations if the neural network included no non-linear operations, such as activation functions like ReLU. In this scenario, a single linear function would be produced by simply multiplying the weight matrices from each layer together. Given that linear functions have well-understood mathematical features, analyzing the behavior of such a system would be considerably simpler. As a result, it would be easy and computationally efficient to compute boundaries on the output, which would greatly simplify verification. But to capture intricate patterns, real-world networks mostly rely on non-linearities, which creates difficulties for analysis and robustness verification. Presence of ReLU or any other nonlinear function makes it difficult to do so, hence we linearize ReLU by using linear bounds as outlined in Fig. 2.

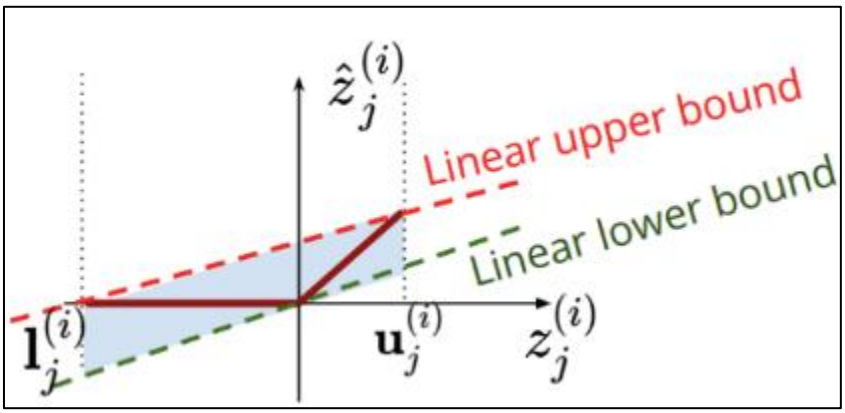



**Figure 2** Lower and Upper bounds on ReLU (Adapted from https://arxiv.org/pdf/1811.00866 )

The main formula is to propagate a linear lower bound for the output neuron with respect to the hidden or input neuron. If we keep linearizing ReLU for the neural network in Fig.1 and keep propagating the lower bounds of the ReLU we get the following equation:

$$f(x) \geq w^{(3)T} D^{(2)} W^{(2)} D^{(1)} W^{(1)} x + \text{const} \quad \ldots\ldots \quad (4)$$

Here D is a diagonal matrix obtained due to computation of lower bounds for ReLU. Equation (4) can be written as:

$$f_{\text{CROWN}}(x) = a^T_{\text{CROWN}} x + c_{\text{CROWN}} \quad \ldots\ldots\ldots \quad (5)$$

Here $a_{\text{CROWN}}$ and $c_{\text{CROWN}}$ are functions of Neural Network weights, and can be computed efficiently on GPUs in a backward manner. The final lower bound can computed by finding minimum of equation (5) which can be represented as follows:

$$f^*_{\text{CROWN}} = \min_{x \in C} a^T_{\text{CROWN}} x + c_{\text{CROWN}} \quad \ldots\ldots\ldots\ldots (6)$$

This completes the CROWN algorithm. Fig. 3 gives a diagrammatic representation of the global lower bound obtained. The illustration helps visualize how the linear bounds approximate the network's behavior under input perturbations. It also highlights the regions where the CROWN method provides tight guarantees. Such visual representations are useful for understanding the strength and limitations of the bound in practical settings.

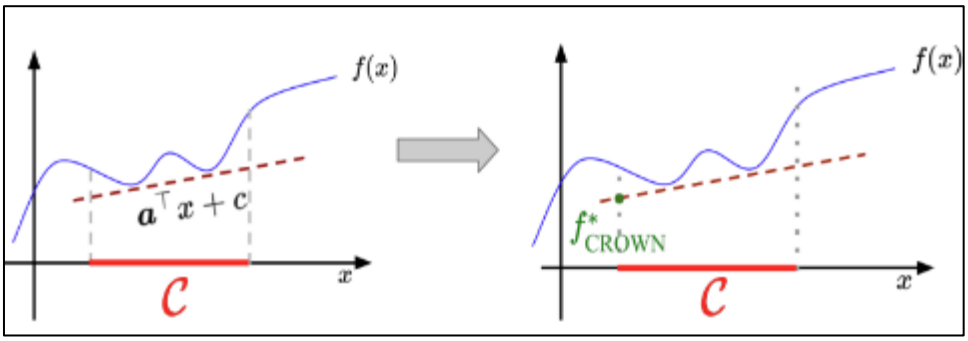



**Figure 3** Global lower bound with Crown (Adapted from https://arxiv.org/pdf/2103.06624 )

*3.1.2. Beta-Crown*

When applied as a stand-alone verification technique, the CROWN method is frequently insufficient to manage large-scale neural networks. This is due to the fact that the constraints generated by CROWN become less relevant in evaluating resilience as the model size grows. The branch-and-bound approach is incorporated into the procedure to overcome this constraint. This method allows for more precise and stringent boundaries by methodically exploring various areas of the input space. Combining these two approaches allows us to balance precision and scalability, increasing the reliability of the verification even for intricate models. We add branch and bound to improve the bounds and make them tighter. To put branch and bound we split a ReLU into two sub-parts where:

$$C_1 = \{x \in C \mid z^{(i)}_j \geq 0\}$$

$$C_2 = \{x \in C \mid z^{(i)}_j \leq 0\}$$

$z^{(i)}_j$ is an unstable ReLU neuron in$C$ but now becomes linear for each sub-part. If the lower bound produced for sub-part $C_i$ ( denoted by $f_{ci}$) is greater than 0, $C_i$ is verified; otherwise, we further branch over domain $C_i$ by splitting another unstable ReLU neuron. The process terminates when all subdomains are verified. The completeness is guaranteed when all unstable ReLU neurons are split.

In order to use branch and bound, [10] these split constraints have to be incorporated into bound propagation, which cannot be handled by the Crown. These split constraints can be represented in terms of a Lagrangian multiplier which can then be propagated backwards. Therefore, the beta factor is introduced which comes from propagation of these

Lagrangian multipliers. Fig.4 shows the comparison between the bounds when the crown is standalone versus when Beta-Crown is used. The method Beta-Crown gives tighter bounds.

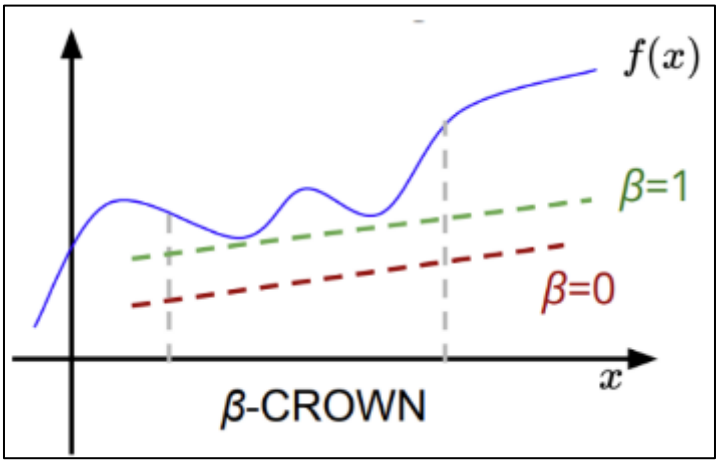



**Figure 4** Global lower bound with Beta-Crown (Adapted from https://arxiv.org/pdf/2103.06624 )

### 3.2. ERAN Analysis

ERAN (ETH Robustness Analyzer for Neural Networks) is a verification tool designed to formally assess the robustness and safety properties of neural networks. It supports diverse network architectures and activation functions, including ReLU and sigmoid. ERAN leverages abstract interpretation techniques, employing mathematical abstractions such as zonotopes and polyhedra to efficiently over-approximate the output sets under input perturbations. This approach enables the tool to provide sound guarantees about network behavior in scenarios where exact analysis is computationally prohibitive. To enhance verification precision, ERAN integrates a branch-and-bound framework that iteratively refines output bounds. Additionally, it can formulate the verification problem as a Mixed Integer Linear Programming (MILP) task for cases demanding exact computations. The tool is compatible with networks developed in popular frameworks like TensorFlow and PyTorch, facilitating broad applicability. By combining these methods, ERAN balances accuracy and scalability, making it a valuable asset for analyzing moderately large networks in safety-critical applications such as autonomous systems and healthcare.

For this verifier, abstract interpretation for neural networks was adopted. Abstract interpretation is a theory which dictates how to obtain sound, computable, and precise finite approximations of potentially infinite sets of behaviors. This verifier subsumes the concepts of the AI2 [8] verifier which is based on an abstraction interpretation method. The method begins by defining abstract transformers for the different kinds of neural network layers. These layers broadly comprise convolutional layers, max pooling layers, and fully connected layers. Using these transformers, the algorithm proves the robustness properties of neural networks by verifying all the points obtained corresponding to the outputs with the same classification.

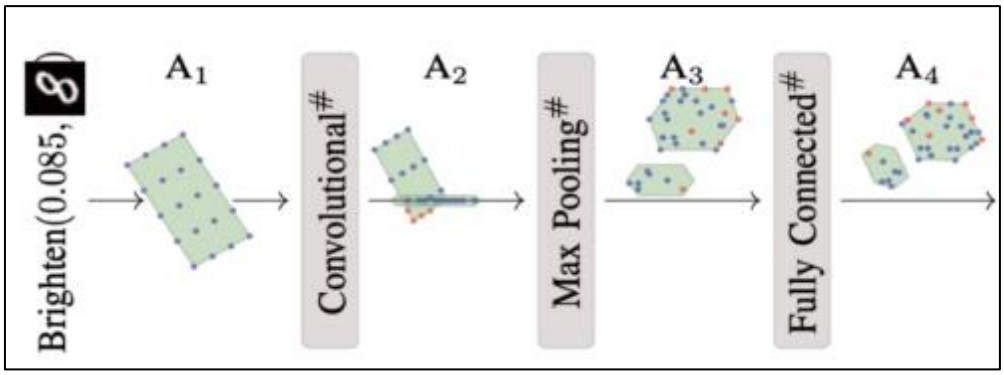



**Figure 5** Abstraction algorithm (Adapted from https://www.cs.utexas.edu/~swarat/pubs/sp2018-ai2.pdf )

The algorithm for ERAN goes like this: AI2 checks that all perturbed inputs are classified the same way. AI2 first creates an abstract element $A1$ capturing all perturbed images. (Here, we use a 2-bound set of zonotopes.) It then propagates

$A1$ through the abstract transformer of each layer, obtaining new shapes. Finally, it verifies that all points in $A4$ correspond to outputs with the same classification. The algorithm is outlined in Fig. 5.

*3.2.1. Two Layer Feed Forward Neural Network*

To analyze the operations on a simple two-layer feed forward neural network layer, there can be in two steps. The steps are explained as follows:

- **Step 1: Compute effect of affine transform**

An affine transformation is a fundamental geometric operation that combines linear transformation with translation. It can be represented as a function that maps coordinates from one space to another while preserving the essential geometric structure of the objects within that space. Specifically, it maintains the collinearity of points, meaning that points lying on a straight line before transformation will continue to lie on a straight line afterward. Additionally, planes remain planes, and sets of parallel lines remain parallel under affine transformation. Certain crucial links are maintained even if affine transformations may change the scale or orientation of geometric shapes. For example, midpoints of line segments and the ratios of distances along parallel lines are unaffected, even when absolute distances and angles are not always maintained. Since the structural integrity of geometric configurations must be maintained during transformations, affine transformations are particularly useful in fields like computer vision, image registration, robotics, and graphics. Mathematically, an affine transformation applies a linear mapping followed by a translation. In two or three dimensions, this can be expressed using matrix operations, where a point is first subjected to a linear transformation defined by a matrix and then shifted by a fixed translation vector. The preservation of linearity and parallelism underpins the robustness of affine models in a variety of engineering and computational contexts.

$$\text{Affine} \equiv (\hat{a} = 0.2\hat{n} + 0.4\hat{m}) \wedge (\hat{b} = 0.1\hat{n} + 0.4\hat{m}) \quad \text{.....(7)}$$

Here, $\hat{n}$ and $\hat{m}$ represent abstract neurons corresponding to the concrete neurons nnn and mmm, respectively. These abstract neurons are typically derived through an abstraction process that captures essential properties or behaviors of their concrete counterparts while suppressing irrelevant low-level details. In the context of affine transformations, the vectors $\hat{a}$ and $\hat{b}$ are expressed as linear combinations of $\hat{n}$ and $\hat{m}$, illustrating how the abstraction maintains a structural relationship between neuron activations while enabling analysis at a higher semantic level.

- **Step 2 : Compute effect of ReLU**

The Rectified Linear Unit (ReLU) is a widely used piecewise linear activation function in neural networks. It operates by outputting the input value directly if it is positive; otherwise, it outputs zero.

$$Y = \mathrm{ReLU}(x) = \max(0, x) \quad \text{.....(8)}$$

This simple yet effective function introduces non-linearity into the model, allowing neural networks to learn complex patterns while maintaining computational efficiency. ReLU is favored in deep learning architectures because it mitigates the vanishing gradient problem that commonly affects sigmoid and tanh activation functions, enabling faster training convergence. Additionally, its sparsity-inducing behavior—where negative inputs are set to zero—can lead to more efficient representations by activating only a subset of neurons.

Using the above two steps, the abstract transformation of ReLU can be given by the following equations:

$$f^{\#}_{\mathrm{ReLU}} = ReLU^{\#}_2(b) \circ ReLU^{\#}_1(a)(Affine) \quad \text{.....(9)}$$

$$ReLU^{\#}_i(x_i)(\psi) = (\psi \sqcap \{x_i \geq 0\}) \sqcup \psi_0 \quad \text{..... (10)}$$

$$\psi_0 = \begin{cases} \|x_i = 0\|(\psi), & \text{if } (\psi \sqcap \{x_i < 0\}) \neq \bot \\ \bot, & \text{otherwise} \end{cases} \quad \text{.....(11)}$$

Here, $f^{\#}_{ReLU}$ is the composition of the abstract transformation of the $ReLU^{\#}_1(a)$ and $ReLU^{\#}_2(b)$ taking into consideration the affine transformation computed in step 1. Both the steps can have combined diagrammatic representation as shown in Fig. 6.

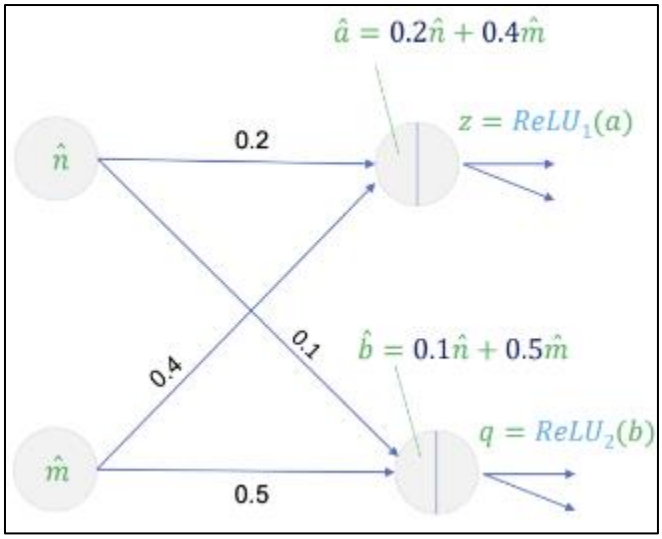



**Figure 6** Representation of affine transformation

To check the robustness property of the neural network we use the definition below. A robustness property for a neural network $N : R^m \to R^n$ is a pair $(X, C) \in \mathcal{P}(R^m) \times \mathcal{P}(R^n)$ consisting of a robustness region *X* and a robustness condition *C*. We say that the neural network *N* satisfies a robustness property ( , *C*) if $N(\bar{x}) \in C$ for all $\bar{x} \in X$.

*3.2.2. Robustness Verification Steps*

To verify the robustness, we divide the algorithm into two steps:

**Step 1** Define the adversarial region around x based on the perturbation of interest (brightness, $L_\infty$, rotations, etc) This is explained using an example which can be represented by the equation (12).

$$L_\infty Ball : Ball_\varepsilon(x) = \{y | \|x - y\|_\infty < \varepsilon\} \quad \text{.....} \quad (12)$$

**Step 2** Attack tries to find image y in the region where *N N* ($x$) ≠ *N N* ($y$). This can be represented by Fig. 7.

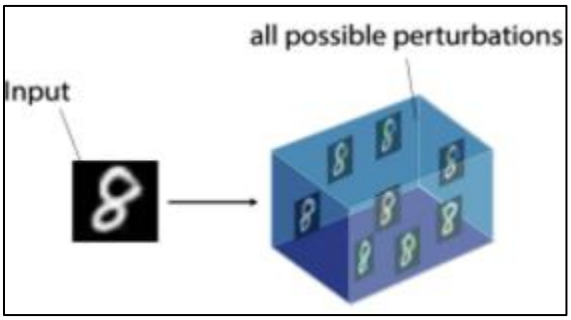



**Figure 7** Representation of perturbation region (Adapted from https://github.com/sarathknv/adversarial-examples-pytorch )

The goal is to prove step 2 never succeeds. If all the concrete inputs lie inside the adversarial region, then we can say that the neural network is robust to the input perturbations else the neural network cannot verify the inputs.

## 4. Empirical Results and Discussion

For experimental purposes, we tested our verifiers and carried out comparative analysis on two datasets MNIST and CIFAR-10. The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. This dataset comprises 100 small square 28*28-pixel gray-scale images of handwritten single digits between 0 and 9. The CIFAR-10 dataset consists of 100, 32*32 color images in 10 classes. For testing and analysis, the two verifiers

were tested on different types of convolutional neural networks such as ConvSmall, ConvBig, and ResNet18 [6]. The framework description of the models for the respective datasets is given in Table 1.

**Table 1** Neural network model description

| Model Name | Model Description |
|---|---|
| ConvSmall (MNIST) | Conv (1, 16, 4) – Conv (16, 32, 4) – Linear (800, 100) – Linear (100, 10) |
| ConvBig (MNIST) | Conv (1, 32, 3) – Conv (32, 32, 4) – Conv (32, 64, 3) – Conv (64, 64, 4) – Linear (3136, 512) – Linear (512, 512) – Linear (512, 10) |
| ConvSmall (CIFAR-10) | Conv (3, 16, 4) – Conv (16, 32, 4) – Linear (1152, 100) – Linear (100, 10) |
| ConvBig (CIFAR-10) | Conv (3, 32, 3) – Conv (32, 32, 4) – Conv (32, 64, 3) – Conv (64, 64, 4) – Linear (4096, 512) – Linear (512, 512) – Linear (512, 10) |
| ResNet18 (CIFAR-10) | 18-layer deep CNN |

We conducted a comparative analysis of our two verifiers, Beta-Crown and ERAN, across five distinct neural network architectures. ConvBig and ConvSmall were evaluated on both the MNIST and CIFAR-10 datasets, while ResNet18 was tested specifically on CIFAR-10.

In terms of verification accuracy, Beta-Crown generally outperformed ERAN for ConvBig and ConvSmall networks on both MNIST and CIFAR-10. However, for the significantly larger ResNet18 on CIFAR-10, ERAN demonstrated superior accuracy.

When considering the total time required for verification, Beta-Crown consistently showed much better performance, considerably faster than ERAN across all datasets and networks. Further, we observed differences in how accuracy is scaled with network size for each verifier. For ERAN, accuracy for ConvBig notably decreased compared to ConvSmall within the same datasets. Conversely, Beta-Crown's accuracy tended to increase for larger neural networks when compared to smaller ones.

In summary, our implementation indicates that Beta-Crown is generally more efficient and effective at verifying the robustness properties of neural networks than ERAN, particularly in terms of speed and its performance scaling with increasing network size for the architectures where it excelled. The above comparison can be tabulated as in Table 2 and Table 3.

**Table 2** ERAN performance evaluation

| ERAN | ConvBig (MNIST) | ConvSmall (MNIST) | ConvBig (CIFAR-10) | ConvSmall (CIFAR-10) | ResNet18 (CIFAR-10) |
|---|---|---|---|---|---|
| Total verified | 35/95 | 42/100 | 21/60 | 31/70 | 83/100 |
| Unsafe | 0/95 | 0/100 | 0/60 | 0/70 | 0/83 |
| Computational time in seconds | 2176.80 | 85.18 | 4130.83 | 130.75 | 12991.59 |
| Accuracy | 36.84 | 42 | 35 | 44.28 | 83 |

**Table 3** Beta-Crown performance evaluation

| Beta - Crown | ConvBig (MNIST) | ConvSmall (MNIST) | ConvBig (CIFAR-10) | ConvSmall (CIFAR-10) | ResNet18 (CIFAR-10) |
|---|---|---|---|---|---|
| Total verified | 81/100 | 79/100 | 53/100 | 46/100 | 52/100 |
| Unsafe | 18/100 | 21/100 | 43/100 | 51/100 | 31/100 |
| Computational time in seconds | 183.49 | 66.37 | 180.73 | 121.29 | 181.90 |
| Accuracy | 81 | 79 | 53 | 46 | 52 |

## 5. Technical Challenges and Solutions

Implementing and ensuring interoperability between the chosen verifiers, ERAN and Beta-Crown, presented several technical challenges, primarily related to dependency management and data format standardization. Overcoming these hurdles was crucial for conducting a fair and accurate comparative analysis.

### 5.1. ERAN Verifier Implementation Hurdles

The initial phase of integrating the ERAN verifier involved successfully cloning its GitHub repository onto our server. However, a significant obstacle arose during the subsequent dependency installation process. Automated generation of necessary related folders, a standard expectation for such installations, consistently failed. This issue necessitated a deeper investigation and independent research to identify a viable solution. Our findings revealed that the required directory structure was not being created programmatically. Consequently, we adopted a manual approach:

- **Manual Folder Creation:** We proactively created the essential folders that the installation process failed to generate automatically.
- **Dataset Acquisition and Upload:** The datasets required for ERAN's operation were not automatically downloaded or linked during the installation. We therefore had to manually download these necessary datasets and securely upload them to our server's designated directories.

Upon implementing these manual interventions, we successfully resolved the dependency issues, enabling the complete and functional implementation of the ERAN verifier on our system.

### 5.2. Dataset Format Standardization for Comparative Analysis

A second critical challenge emerged from the disparate dataset formats utilized by Beta-Crown and our specific requirements for comparative analysis. Beta-Crown's expected dataset structure adhered to the guidelines stipulated by the Verification of Neural Networks (VNN) competition. This VNN-competition-specific format differed fundamentally from the dataset format we had prepared and standardized for our experiments and for use with the ERAN verifier.

To ensure a consistent and equitable comparison between Beta-Crown and ERAN, it was imperative that both verifiers operate on identically formatted input data. This required us to develop a robust methodology for converting the VNN-competition-compliant dataset format into our internal standardized format. This conversion process involved:

- **Format Discrepancy Analysis:** Thoroughly analyzing the structural and content differences between the VNN competition format and our target format.
- **Custom Data Transformation:** Developing custom scripts or routines to programmatically convert the datasets, ensuring all necessary features and labels were accurately mapped and preserved.

Successfully aligning the dataset formats was a pivotal step, enabling us to perform a direct and meaningful comparative analysis of both verifiers under consistent experimental conditions.

## 6. Future work

The insights gained from this comparative analysis open several promising avenues for future research, extending the capabilities of both ERAN and Beta-Crown algorithms. Our future work concerns deeper analysis of particular mechanisms, exploration of novel methodological proposals, and investigation driven by scientific curiosity. The following ideas could be tested to yield improved results and potentially introduce novel contributions to the field:

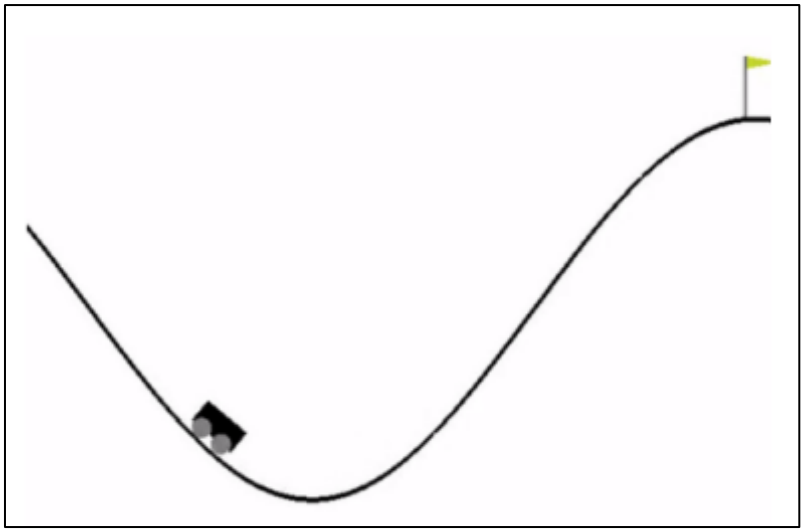
One significant area of exploration involves refining the bounding techniques. For ERAN, the current methodology often relies on relatively loose bounds for over-approximation. Future efforts could investigate how incorporating tighter bounding approaches, similar to those successfully employed by Beta-Crown, could lead to more efficient and precise verification results for ERAN. This cross-pollination of techniques holds the potential to significantly enhance ERAN's performance.

Conversely, Beta-Crown's branch and bound strategy presents an opportunity for further optimization. Currently, during the branch and bound process, lower bounds are computed for both the linear and negative regions of the Rectified Linear Unit (ReLU) activation function. Given that the linear region intrinsically yields tighter bounds, an extended approach could be to utilize calculations based on the linear region, potentially omitting or minimizing the computation for the negative region. This targeted refinement could lead to substantial computational savings without compromising the tightness of the overall bounds, thereby yielding a more efficient verification process for Beta-Crown.

Beyond methodological improvements within existing algorithms, the current work primarily demonstrates robust verification for image classification tasks. A crucial generalization of this research involves extending these verification guarantees to other critical domains, such as robust deep reinforcement learning. This expansion is vital for deploying AI systems in safety-critical applications where reliable decision-making under adversarial conditions is paramount. As a concrete example, the well-known mountain car problem in reinforcement learning could serve as a valuable case study to test and demonstrate the applicability of these robust verification techniques in a dynamic, control-oriented environment. This would pave the way for verifying the robustness of policies learned by deep reinforcement learning agents against various input perturbations or environmental uncertainties.

### 6.1. Mountain Car Problem

The Mountain Car problem is a classic control task in reinforcement learning, serving as an excellent benchmark for agents learning to achieve a goal in a dynamic environment. The core problem statement involves a car initially situated in a valley between two hills (outlined in Fig. 8). The objective for the car is to reach a designated flag positioned at the apex of the right-hand hill. A key challenge of this environment is that the car's engine power is insufficient to directly ascend the hill by simply accelerating forward from its starting position. To successfully reach the target, the car, acting as our reinforcement learning agent, must intelligently build momentum. This is achieved by swinging itself back and forth between the two hills, gradually accumulating enough velocity to finally surmount the peak. The agent interacts with its environment by selecting discrete actions. For each chosen action, the car's state is updated, which is comprehensively defined by its current position and velocity. At any given state, the car has a limited set of three possible actions: it can accelerate forwards, accelerate backwards, or remain idle (do nothing).Within the scope of our project, we successfully trained a neural network to control the car to achieve the goal of reaching the top of the hill. A significant extension of this work, and a crucial direction for future research, would be to formally verify the robustness of this trained neural network. This would involve ensuring that the car's learned policy remains effective and achieves its goal even when faced with small perturbations in its sensory inputs (position and velocity) or in the environment itself.



https://gymnasium.farama.org/environments/classic_control/mountain_car/ )

**Figure 8** Mountain car problem representation (Adapted from

## 7. Conclusion

This study presents a comprehensive comparative analysis of two leading neural network verification tools—Beta-CROWN and ERAN—focusing on their effectiveness in certifying the robustness of convolutional neural networks against adversarial perturbations. The results demonstrate that Beta-CROWN consistently outperforms ERAN in terms of computational efficiency and verification accuracy, especially for deeper networks like ConvBig on MNIST and CIFAR-10 datasets, while ERAN shows stronger performance for larger architectures like ResNet18. The use of tighter linear bounds in Beta-CROWN and abstract interpretation in ERAN highlights the trade-offs between scalability and precision. Overall, Beta-CROWN proves to be more adaptable for larger models, whereas ERAN remains a reliable choice for precision-critical applications. This comparative benchmarking not only aids practitioners in selecting suitable verification tools for specific architectures but also sets the stage for future improvements in hybrid verification approaches. By enabling the deployment of provably robust neural networks in safety-critical fields such as autonomous driving and healthcare, this study contributes to societal trust in AI systems; moving forward, expanding verification techniques to cover reinforcement learning and control systems will further strengthen AI safety in real-world applications.

## References

[1] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu.2017. Safety verification of deep neural networks. In Computer Aided Verification, 29th International Conference (CAV),pages 3– 29.

[2] V. Tjeng, K. Xiao, and R. Tedrake. 2019. Evaluating robustness of neural networks with mixed integer programming.In International Conference on Learning Representations (ICLR), 2019.

[3] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. 2017. Reluplex: An efficient smt solver for verifying deep neural networks.In International Conference on Computer Aided Verification (CAV).

[4] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh and Luca Daniel.2018.Efficient Neural Network Robustness Certification with General Activation Functions. arXiv:1811.00866v1 [cs.LG] 2 Nov 2018 .

[5] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana.2018. Efficient formal safety analysis of neural networks. In Advances in Neural Information Processing Systems (NeurIPS).

[6] Kaidi Xu, Zhouxing Shi2, Huan Zhang3, Yihan Wang, Kai-Wei Chang , Minlie Huang , Bhavya Kailkhura , Xue Lin1 and Cho-Jui Hsieh.2020. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. arXiv:2002.12920v3 [cs.LG] 26 Oct 2020.

[7] Luca Pulina and Armando Tacchella.2010. An abstraction-refinement approach to verification of artificial neural networks. In Computer Aided Verification, 22nd International Conference (CAV).

[8] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI2 : Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In IEEE Symposium on Security and Privacy, 2018.

[9] R. Ehlers.2017. Formal verification of piece-wise linear feed-forward neural networks. In the International Symposium on Automated Technology for Verification and Analysis (ATVA).

[10] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and Zico Kolter.2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification.arXiv:2103.06624v2 [cs.LG] 31 Oct 2021.

[11] R. R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda.2018. A unified view of piecewise linear neural network verification. In Advances in Neural Information Processing Systems (NeurIPS).

[12] R. Bunel, A. De Palma, A. Desmaison, K. Dvijotham, P. Kohli, P. H. S. Torr, and M. P. Kumar.2020. Lagrangian decomposition for neural network verification. Conference on Uncertainty in Artificial Intelligence (UAI).

[13] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara Navidpour. Proving programs robust. 2011. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE), pages 102–112.

[14] Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker.2015. Towards verification of artificial neural networks. In Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV).

[15] Rupak Majumdar and Indranil Saha. 2009. Symbolic robustness analysis. In Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS), pages 355–363.