Verification of Neural Networks Competition

Victor Gherghel, Marian Ostate

February 3, 2024

Abstract

We choosed the Vision Transformer (ViT) model for our project, drawn to its innovative approach and promising capabilities. Our decision to choose ViT comes from a convergence of factors that align perfectly with the distinctive requirements and targets of our project. The big interest and ongoing support within the research community for ViT helped us with making our project decision. This ensures that we are working with a model backed by many collective knowledge. The purpose of this project is to reproduce and understand the results from the VNN-COMP 20231 for ViT benchmark. In summary, our choice of ViT is influnced by a colection of its unique features and adaptability to project requirements.

1 Introduction

The Verification of Neural Networks Competition, focuses on the formal verification of neural networks, aiming to address the challenges associated with verifying the correctness and safety of neural network models. Verification in this context involves ensuring that a neural network behaves as intended and does not exhibit undesirable or unsafe behavior.

In competitions like VNN-COMP, benchmarks serve as a testing ground, resolving a mix of neural network challenges from verification tools. These challenges span different applications, like image recognition (like in the case of Vision Transformer) including changes that are made to the input data of a neural network with the intention of causing the network to make a mistake.

In this paper our first chosen tool, $\alpha, \beta-$**CROWN** focuses on ensuring the robustness of neural networks against adversarial examples, whereas the second chosen tool, **PyRat**, employs formal methods to rigorously check the safety and correctness of neural networks, preventing unexpected behaviors. Also, we have chosen the **Vision Transformer (ViT)** benchmark/dataset which includes a lot of neural network architecture for computer vision tasks, transforming image data into sequences for efficient processing. Together, these tools combined with the benchmark/dataset contribute to the development of reliable and secure neural networks, addressing challenges in robustness and safety verification.

2 Dataset

Vision Transformer (ViT) can be evaluated and verified using the VNNLIB (Verified Neural

Network Library) and ONNX (Open Neural Network Exchange) verification libraries and tools that are used for different aspects:

- **VNNLIB (Verified Neural Network Library)**: VNNLIB is typically associated with formal verification tasks because it provides a standardized format for specifying neural network properties and constraints, making it easier for verification tools to assess the correctness and safety of neural networks.
- **ONNX (Open Neural Network Exchange)**: ONNX is a more general-purpose format for representing neural network models because it allows models to be exchanged between various frameworks and tools, facilitating interoperability between different deep learning frameworks.

Also, Vision Transformer (ViT) dataset, uses 100 instances of `onnx/pgd` libraries and 100 `onnx/ibp` libraries which refers to specific types of adversarial examples generated for testing the robustness of ViT models:

- **onnx/pgd (Projected Gradient Descent)** is an iterative optimization method where perturbations are added to input data to create adversarial examples, aiming to maximize misclassification.
- **onnx/ibp (Interval Bound Propagation)** is a technique that establishes bounds on the output of neural networks, contributing to the generation of robust adversarial examples.

In summary, the use of VNNLIB and ONNX in the context of Vision Transformer would likely involve formal verification tasks (using VNNLIB) and model interchangeability (using ONNX) for experimentation and evaluation purposes.

3 Tools

3.1 $\alpha, \beta-$CROWN

$\alpha, \beta-$CROWN functions as a neural network verifier employing a streamlined linear bound propagation framework and branch-and-bound techniques. Its computational efficiency is enhanced when deployed on GPUs, allowing effective scaling to sizable convolutional networks with millions of parameters. The alpha-beta-CROWN method is capable of offering provable assurances of robustness against adversarial attacks while also verifying other general properties inherent in neural networks. [1]

1. CROWN is a fast verification method that uses a backward propagation of linear inequalities through a neural network, relaxing activation functions with linear bounds.
2. $\alpha-$CROWN optimizes CROWN for the Fast-and-Complete verifier, improving intermediate and final layer bounds through a variable α. It's more powerful than LP due to its ability to refine intermediate layer bounds inexpensively.

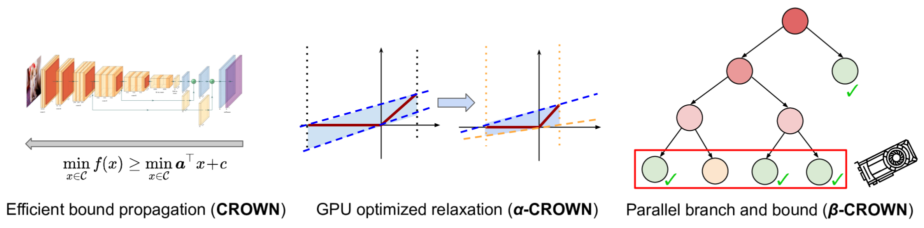

$\min_{x\in\mathcal{C}} f(x) \geq \min_{x\in\mathcal{C}} \boldsymbol{a}^\top x + c$
Efficient bound propagation (**CROWN**)
GPU optimized relaxation (***α*-CROWN**)
Parallel branch and bound (***β*-CROWN**)

Figure 1: ViT Supported Features

3. $\beta-$CROWN introduces an optimizable parameter β to include split constraints in branch and bound into the CROWN process. This creates a robust and scalable neural network verifier by combining branch and bound with efficient GPU-accelerated bound propagation.

Installation and Usage of $\alpha, \beta-$CROWN

```
# Create directory and clone repositories

mkdir abc_tmp
cd abc_tmp
git clone \href{https://github.com/Verified-Intelligence/alpha-beta-CROWN.git}
git clone %\url{https://github.com/Verified-Intelligence/auto_LiRPA.git}% in abcrown/au
git clone %\url{https://github.com/ChristopherBrix/vnncomp2023_benchmarks.git}% in abc_t

cd vnncomp2023_benchmarks ; ./setup.sh -

# which will unzip all the vnnlib and onnx archives

# Clone and install miniconda (conda will also work)

cd abc_tmp
install miniconda + all the features
mkdir -p ~/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/minicon
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm -rf ~/miniconda3/miniconda.sh

# restart console - otherwise it will not work

# Create the alpha-beta-crown environment

cd abcrown/complete verifier
# Remove the old environment, if necessary.
conda deactivate; conda env remove --name alpha-beta-crown

# Install all dependencies into the alpha-beta-crown environment
```

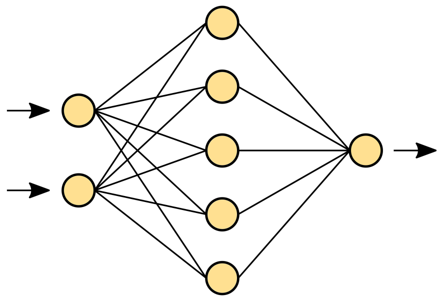
```
conda env create -f complete_verifier/environment.yaml --name alpha-beta-crown

# Activate the environment

conda activate alpha-beta-crown
cd complete_verifier

# Try cifar_resnet_2b.yaml to see if alpha-beta-crown is working properly

# Because it is a VM with no dedicated GPU, in order to bypass the "Not enough memory"
# or "No CUDA GPUs available" we must use the \textbf{--device cpu} flag
python abcrown.py --config ./exp_configs/vnncomp23/cifar_resnet_2b.yaml --device cpu

# Result: safe incomplete in 3.5271 seconds, check Cifar_resnet_2b results.txt in

%\url{https://github.com/VictorGhe/VF-proiect/blob/main/alpha-beta-crown/}%

# Try vit.yaml to see if alpha-beta-crown is working properly
python abcrown.py --config ./exp_configs/vnncomp23/vit.yaml --device cpu

# Results: check "VITresults", "VITresults#2", "VITresults#3" in
%\url{https://github.com/VictorGhe/VF-proiect/tree/main/alpha-beta-crown}%

# Proofs can be checked at the link below
All the pictures and extra useful documents can be found at
https://github.com/VictorGhe/VF-proiect/tree/main/alpha-beta-crown/proofs

# Results after the execution can be checked at the link below:
https://github.com/VictorGhe/VF-proiect/tree/main/alpha-beta-crown/results
```

3.2 PyRat

PyRAT (Python Reachability Assessment Tool) is an open source tool used to examine a deep neural network's state reachability. Its foundation is abstract interpretation, a method for approximating programme execution routes in order to reason the programme behaviour. PyRAT seeks to determine if a neural network can achieve any of the goal states from any given input by using a collection of target states and the neural network as input. [2]
PyRAT allows users to define safety properties, assess output, and improve accuracy by dividing input into many layers for a variety of neural network types. PyRAT can efficiently work on both CPU and GPU. PyRAT is accurate since the output bounds attained are consistently a reliable estimate of the actual outcomes. Multiple verification modes and domains can be chosen, depending on the benchmark. PyRAT will combine local counterexample testing and GPU computing on a broader network to confirm or deny a property. Its performance was demonstrated in the VNN-COMP 2023 competition placing on the 3rd place.

PyRAT's uses: [3]

1. tool for safety testing, since it can confirm whether a neural network is not exhibiting any unwanted behaviours, such crashing or giving inaccurate outputs.
2. analysis of robustness: PyRAT may be used to examine how resilient a neural network is against hostile inputs.
3. explainability: The decisions made by a neural network may be explained using PyRAT.



Practical examples of uses include confirming that a neural network used for medical diagnosis does not generate false diagnoses, examining how resilient a neural network is to hostile inputs, and explaining the reasoning behind a neural network's image decision-making.

Key characteristics of PyRAT [4] make it stand out as a powerful tool for neural network research. It's wide support for a variety of neural network topologies, including feedforward, convolutional, recurrent, and deep reinforcement learning networks, is its first noteworthy characteristic. PyRAT's adaptability to the difficulties included in different model designs is ensured by its flexibility. PyRAT's exceptional accuracy in determining whether a neural network can achieve a given state is another noteworthy feature that sets it apart from the competition and gives users trustworthy information about the capabilities and constraints of their models. Moreover, PyRAT stands out for its efficiency, especially with big neural networks. This efficacy not only simplifies the analysis procedure but also establishes PyRAT as a useful tool for practitioners and academics dealing with intricate and large-scale models. To summarise, PyRAT is an effective tool for neural network analysis because of its accuracy, efficiency, and support for a variety of designs.

Installation and Usage of PyRAT - unsuccessful

```
# Create directory and clone repositories

mkdir pyrat
cd pyrat
git clone https://github.com/pyratlib/pyrat.git

# Create vnncomp benchmark and unzipping files
git clone https://github.com/ChristopherBrix/vnncomp2023_benchmarks.git
```

```
cd vnncomp ; ./setup.sh -

# which will unzip all the vnnlib and onnx archives

cd pyrat
./install_tool.sh

For running, the following command was intended to be used. It writes results in out.cs
./run_all_categories.sh v1 /home/parallels/pyrat/vnn_config/ . ./out.csv ./counterexamp

# Proofs can be checked at the link below
All the pictures from https://github.com/VictorGhe/VF-proiect/tree/main/pyrat/proofs
```

4 Experimental Results

The execution of $\alpha, \beta-$CROWN and PyRat tools on the Vision Transformer (ViT) dataset are called Instance Runtimes, presenting detailed insights into the computational performance. These results quantify the time taken for individual instances during the verification process, offering a technical perspective on the efficiency of the verification tools. Also, the Instance Runtimes provide information for assessing the computational complexity and scalability of $\alpha, \beta-$CROWN and PyRat in analyzing Vision Transformer models.

All the results of the $\alpha, \beta-$CROWN execution on the Vision Transformer (ViT) dataset are documented and can be accessed at the following GitHub repository link: `https://github.com/VictorGhe/VF-proiect/tree/main/alpha-beta-crown/results`. Within this repository, a detailed CSV file containing the obtained results and the command execution outputs are also available in the same designated folder for further reference and analysis.

Also, comprehensive documentation, inclusive of detailed results and challenges encountered, were stored in the following GitHub repository `https://github.com/VictorGhe/VF-proiect`.

5 Conclusions

References

[1] Alpha-beta-crown, 2023. URL: `https://github.com/Verified-Intelligence/alpha-beta-CROWN`, Accessed on: 15.01.2024.

[2] Pyrat, 2023. URL: `https://github.com/ChristopherBrix/vnncomp2023_results/tree/main/pyrat`, Accessed on: 15.01.2024.

[3] Pyrat, 2023. URL: `https://git.frama-c.com/pub/pyrat`, Accessed on: 15.01.2024.

[4] Pyrat-analyzer, 2023. URL: `https://pyrat-analyzer.com/`, Accessed on: 15.01.2024.