Kalpa Publications in Computing
Volume 16, 2023, Pages 47–58
Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems

Supporting Standardization of Neural Networks Verification with VNN-LIB and CoCoNet

Stefano Demarchi[1], Dario Guidotti[2], Luca Pulina[2], and Armando Tacchella[1]

[1] Università degli Studi di Genova, Viale Causa, 13, 16145 Genova, Italy
stefano.demarchi@edu.unige.it armando.tacchella@unige.it
[2] Università degli Studi di Sassari, Via Roma 151, 07100 Sassari, Italy
{dguidotti,lpulina}@uniss.it

Abstract

The interest in the verification of neural networks has been growing steadily in recent years and there have been several advancements in theory, algorithms and tools for the verification of neural networks. Also propelled by VNNCOMP — the annual competition of tools for the verification of neural networks — the community is making steady progress to close the gap with practical applications. In this scenario, we believe that researchers and practitioners should rely on some commonly accepted standard to describe (trained) networks and their properties, as well as a toolset to visualize and to convert from common formats to such standard. The purpose of VNN-LIB and CoCoNet is precisely to provide such standard and toolset, respectively. In this paper we briefly describe the principles and design choices behind the current version of VNN-LIB standard, and we give an overview of the current and planned capabilities of CoCoNet.

1 Introduction

Recently, there has been a significant surge of interest in Machine Learning (ML) and Neural Networks (NNs) in both the research [18, 34, 29] and industrial [35, 22, 16] communities which has been fueled by the successful application of NNs to several tasks in various domains across computer science. However, the application of NNs in domains where safety and security are of paramount importance is still limited due to the lack of formal guarantees on their behavior. The trustworthiness of NNs has been questioned ever since the discovery of issues known as adversarial perturbations [10, 28], where minor variations in the inputs of NNs can result in unforeseeable and undesirable changes in their behavior. As a consequence, the interest in proving the compliance of NNs with properties of interest has been growing steadily together with their increasing popularity.

Current state-of-the-art verification tools can handle NNs whose size remains relatively small when compared to NNs that are popular in practical applications, but noteworthy progress has been achieved in theory, algorithms, and tools for the verification of NNs as witnessed by publications spanning across the past decade [23, 24, 17, 26, 30, 31, 4, 2, 13, 14, 37, 33, 32, 36, 6, 21, 9]. More recently, thanks to the International Verification of Neural Networks

Competition VNNCOMP [5] the community is enjoying an annual update on the state of affairs in verification of NNs, and more researchers are attracted by the challenge of making verification viable also for NNs of practical size. However, the lack of a common standard to describe both (trained) networks and the related properties heavily limits the use of verification tools across different domains, the reproducibility of results and the sharing of benchmarks across the community.

To mitigate these issues we propose the VNN-LIB standard [11] for the description of NNs and related properties and the tool CoCoNet for the visualization and conversion of NNs. A preliminary version of the VNN-LIB format is already adopted by VNNCOMP, but the lack of a commonly accepted format and the wide variety of NN designs featured by applications is still a limiting factor. We wish to create a standard which enables us to collect and organize as many benchmarks as possible in order to make them available to the organizers of VNNCOMP and to the research community at large. We recall that similar efforts, like TPTP [27], SATLIB [15], QBFLIB [19], SMTLIB [3] with their associated competitions, contributed substantially to the progress of Automated Theorem Proving, Satisfiability for Boolean and Quantified Boolean logics and Satisfiability Modulo Theory, respectively. We believe that similar results can be achieved in the field of NN verification, but we must acknowledge that the description of NNs is less standardized and also more complex to provide than, e.g., Boolean formulas in conjunctive normal form as provided by the DIMACS standard featured by SATLIB. This is why, in parallel with the VNN-LIB effort, we decided to develop CoCoNet with three aims: (*i*) to visualize NNs persisted in formats that are common in the machine learning community, e.g., ONNX, TensorFlow, PyTorch, Keras, nnet; (*ii*) to enable users to add properties to NNs using a graphical syntax; (*iii*) finally, to save NNs and associated properties in the VNN-LIB format. In this way, the developers of NN verification technology could focus in supporting just the VNN-LIB standard, but researchers and practictioners in other fields who are interested in verifying their NNs can create inputs for the verification tools without having to know VNN-LIB in detail.

The rest of the paper is structured as follows. In Section 2 we introduce some basic concepts and definitions that we will use in the rest of the paper. In Section 3 we describe the VNN-LIB standard, whereas in Section 4 we present the tool CoCoNet and its current and planned capabilities. We conclude the paper in Section 5 with some final remarks.

2 Background

Verification of Neural Networks. Inputs and outputs of operators are *tensors*, i.e., multidimensional arrays over some domain, usually numerical. If we let $\mathbb{D}$ be any such domain, a k-dimensional tensor on $\mathbb{D}$ is denoted as $x \in \mathbb{D}^{n_1 \times \cdots \times n_k}$. For example, a vector of n real numbers is a 1-dimensional tensor $x \in \mathbb{R}^n$, whereas a matrix of $n \times n$ Booleans is a 2-dimensional tensor $x \in \mathbb{B}^{n \times n}$ with $\mathbb{B} = \{0, 1\}$. A specific element of a tensor can be singled-out via *subscripting*. Given a k-dimensional tensor $x \in \mathbb{D}^{n_1 \times \cdots \times n_k}$, the element $x_{i_1,\ldots,i_k} \in \mathbb{D}$ is a scalar corresponding to the indexes $i_1, \ldots, i_k$. For example, in a vector of real numbers $x \in \mathbb{R}^n$, x_1 is the first element, x_2 the second and so on. In a matrix of Booleans $x \in \mathbb{B}^{n\times n}$, $x_{1,1}$ is the first element of the first row, $x_{2,1}$ is the first element of the second and so on. An *operator* f is a function on tensors $f : \mathbb{D}^{n_1 \times n_h} \to \mathbb{D}^{m_1 \times m_k}$ where h is the dimension of the input tensor and k is the dimension of the output tensor. Given a set $F = \{f_1, \ldots, f_p\}$ of p operators, a *feedforward neural network* is a function $\nu = f_p(f_{p-1}(\ldots f_2(f_1(x)) \ldots))$ obtained through the composition of the operators in F assuming that the dimensions of their inputs and outputs are *compatible*, i.e., if the output of f_i is a k-dimensional tensor, then the input of f_{i+1} is also a

k-dimensional tensor, for all $1 \leq i < p$. Given a neural network $\nu : \mathbb{D}^{n_1 \times n_h} \to \mathbb{D}^{m_1 \times m_k}$ built on the set of operators $\{f_1, \ldots, f_p\}$, let $x \in \mathbb{D}^{n_1 \times n_h}$ denote the input of ν and $y_1, \ldots, y_p$ denote the outputs of the operators $f_1, \ldots, f_p$ — therefore y_p is also the output y of ν. We assume that, in general, a *property* is a first order formula $P(x, y_1, \ldots y_p)$ which should be satisfied given ν. More formally, given p bounded sets $X_1, \ldots, X_p$ in I such that $\Pi = \bigcup_{i=1}^{p} X_i$ and s bounded sets $Y_1, \ldots, Y_s$ in O such that $\Sigma = \bigcup_{i=1}^{s} Y_i$, we wish to prove that

$$\forall x \in \Pi \to \nu(x) \in \Sigma. \tag{1}$$

The definition of the property given in equation (1) consists of a *pre*-condition $x \in \Pi$ and a *post*-condition $\nu(x) \in \Sigma$. The *pre*-condition encodes the bounds of the input space, i.e., bounds the variables that are fed to the network, and the *post*-condition defines the safe zone(s), outside which the verification task fails.

VNN Competition. In 2020, the first VNNCOMP [5] was organized in order to provide researchers and practitioners with a forum to challenge verification tools on benchmarks obtained from applications. VNNCOMP allows researchers to compare their tools on many verification benchmarks and provides a baseline for researchers interested in the field. In these years, the need to standardize the representation of the benchmarks clearly emerged and a preliminary version of VNN-LIB has been considered. The last iteration of the competition [20] saw 11 participants and 12 proposed benchmarks. Thanks to the guidelines provided by VNNCOMP all the benchmarks follow the rules for the preliminary VNN-LIB standard that we describe. In some cases, we also propose restrictions with respect to the allowed architectures, because we decided to focus the current version of the standard on feed-forward networks, and to provide extensions later on.

Open Neural Network Exchange Format. The Open Neural Network Exchange (ONNX) format [1] is an open-source format for representing machine learning models. It allows developers to transfer trained models between different learning frameworks with ease, making it possible to take advantage of the unique features and strengths of each framework. ONNX is supported by a growing number of companies and organizations. The ONNX format is designed to be flexible and extensible, allowing for the representation of a wide range of learning models. It supports a variety of neural network architectures, including convolutional neural networks, recurrent neural networks, and generative adversarial networks. It also supports a variety of data types and is capable of representing complex operations and data transformations. One of the main benefits of this format is its ability to facilitate model interoperability. In theory, by using ONNX, developers can train a deep learning model using one framework and deploy it using another. This allows developers to take advantage of the unique features of each framework without having to rewrite their models from scratch.

Satisfiability Modulo Theory Library. The Satisfiability Modulo Theories Library (SMT-LIB) language [3] is a standard format for expressing logical formulas in Satisfiability Modulo Theories (SMT). SMT is a a field of automated reasoning whose aim is to determine whether a logical formula in a particular decidable subclass of first-order logic is satisfiable, and it can be viewed as an extension of Boolean Satisfiability with, e.g., linear and nonlinear arithmetic, arrays, strings and combinations thereof. The SMT-LIB language provides a standardized syntax for expressing logical formulas in a way that SMT solvers can understand and process. The SMT-LIB language defines a set of standard operators and functions for a variety of logical theories, including arithmetic, bit-vectors, arrays, and sets. It also includes a standardized

Layer type	**Layer**	**ONNX node**
Linear layers	Add	Add
	Sub	Sub
	Matrix Multiplication	MatMul
	Fully Connected	Gemm
	Concatenation	Concat
Activation layers	Rectified Linear Unit	ReLU
	Exponential Linear Unit	ELU
	Continuous differentiable Linear Unit	CELU
	Leaky Rectified Linear Unit	LeakyReLU
	Logistic Unit	Sigmoid
	Hyperbolic Tangent Unit	Tanh
	SoftMax Unit	SoftMax
Convolution layers	Convolutional	Conv
Pooling layers	Average Pooling	AveragePool
	Maximum Pooling	MaxPool
Normalization layers	Batch Normalization	BatchNormalization
	Local Response Normalization	LRN
Dropout layers	Dropout	Dropout
Utility layers	Flatten	Flatten
	Reshape	Reshape
	Unsqueeze	Unsqueeze

Table 1: List of the supported ONNX operators in VNN-LIB, grouped by functionality. This set of operators is sufficient for most of the available benchmarks for sequential networks.

way of expressing quantifiers and assertions, as well as supporting the definition of user-defined functions and data types. It is designed to be easy to read and write, and it is supported by a large number of SMT solvers.

3 VNN-LIB

The purpose of the VNN-LIB [11] standard is to provide an unified format for the description of NNs for verification. To support the widest range of network architectures and related properties, the standard builds on the ONNX format to represent the network models, and on the SMT-LIB language for property specification. In the following, we refer to the aggregate of the model and property representations as the VNN-LIB format.

3.1 Network language

As mentioned above, we select the ONNX format to represent the network in the VNN-LIB format: in particular we selected a subset of the format which allows to represent the majority of the models considered in VNNCOMP so far. To guarantee the generality of the selected subset we also tried to include most of the operators needed to represent the networks collected in the ONNX model zoo[1]. We believe that the benchmarks of the previous VNNCOMP together with the model zoo provide a good reference for the VNN-LIB standard as far as generality is concerned. However, since we also strive to ensure that the semantics of the operators considered is sufficently well understood as to avoid bringing uncertainties in the verification processes, we are not currently supporting all the operators.

In Table 1 we show the supported operators, which are enough to model almost every benchmark provided in the VNNCOMP repositories for sequential networks. As we mentioned some kinds of networks, e.g., residual and recurrent networks together with their main operators, are not currently supported by the standard, but they are scheduled to be integrated in future versions leveraging the support and guidance from the community.

3.2 Property specification

For representing verification properties we rely on the SMT-LIB language which is expressive enough to define both classical robustness and more complex specifications. Using this language, we can link the network model by representing inputs and outputs as variables. We can define both the *pre-* and *post-*conditions at once, by defining sets of constraints on the model variables in order to reproduce the formula in Equation (1). If Σ represents multiple safe zones, it is possible to define them with a disjunction. In this case, the verification should return *True* if and only if all the output conditions are satisfied.

For the sake of clarity, we present an example from an Adaptive Cruise Control (ACC) application [8]. In this context the NN's objective is to replicate the function of an ACC similar to those used in real autonomous cars. The goal of the ACC is to maintain the vehicle at a speed set by the user and possibly adapt the speed considering other vehicles proceeding in front of it. The ACC in [8] has one output, i.e., the acceleration a suggested to the *ego* car, and three inputs: the speed of the *ego* car, the relative speed of the *exo* car and the distance between the two. In [8] three different network architectures are proposed to experiment with, increasing in size and complexity: for the sake of brevity, we build here the *Net0* example consisting of two hidden layers of 20 and 10 neurons each with ReLU activation functions, followed by an output linear layer of dimension 1 without a following activation function. The ACC network acts as a function $\nu : I^3 \rightarrow O^1$ with $I = O = \mathbb{R}$. In this setting we define the property *OutBounds*, i.e., a property which checks that the output acceleration does not exceed the bounds provided by the real production ACC that the network should learn, given by the following equation:

$$\begin{array}{rcl} 0 \leq & x_0 & \leq 50 \\ -50 \leq & x_1 & \leq 50 \\ 0 \leq & x_2 & \leq 150 \\ \\ -3 \leq & y_0 & \leq 1 \end{array} \tag{2}$$

where $(x_0, \ldots, x_2) \in \mathbb{R}^3$ is a sample of the input vector and $y_0 \in \mathbb{R}$ is the corresponding output $y_0 = \nu((x_0, \ldots, x_2))$. This property states that, if the speed of the *ego* car is less than 50, the

[1]https://github.com/onnx/models

relative speed of the *exo* car is in the range $[-50, 50]$ and the distance is less than 150, then the suggested acceleration should be limited in the range $[-1, 3]$. It should be noted that the name of the variables in the SMT-LIB property must agree with the identifiers in the ONNX representation of the network of interest for the corresponding input (output) variables. In the example shown in Equation 2 this means that the identifier of the input tensor of the ONNX model is x, whereas the identifier of the output tensor is y. Clearly, subscripts pinpoint elements of the tensor of interest.

While it should be clear from the example how to represent mono-dimensional tensors, representing multi-dimensional tensors may be slightly more complex. In the current iteration of the standard two different notations are allowed for multi-dimensional tensors: the *matrix* notation and the *unrolled* notation.

Matrix notation. Let $X \in \mathcal{I}$ be an n-dimensional tensor in some generic input domain $\mathcal{I} = \mathbb{I}^{d_1 \times \ldots \times d_n}$. The *matrix* notation represent a specific element $x_{i_1, i_2, \ldots, i_n}$ of the tensor X as X_i_1-i_2-...-i_n, where $i_1, ..., i_n$ are the indexes of the element of interest in the dimensions $d_1, ..., d_n$. To better clarify, if we consider the 1-D tensor $X \in \mathbb{I}^n$, the 2-D tensor $Y \in \mathbb{I}^{n \times m}$ and the 3-D tensor $Z \in \mathbb{I}^{n \times m \times p}$ we will have the following representations:

- X_0, X_1, ..., X_i, ..., X_n;
- Y_0-0, Y_0-1, ..., Y_i-j, ..., Y_n-m;
- Z_0-0-0, Z_0-0-1, ..., Z_i-j-k, ..., Z_n-m-p;

In such representation, Z_i-j-k correspond to the element $z_{i,j,k}$ of the tensor Z. It should be noted that the *matrix* notation is the recommended one, since it provides a clearer representation of the variables of the properties.

Unrolled notation. Let $X \in \mathcal{I}$ be an n-dimensional tensor in some generic input domain $\mathcal{I} = \mathbb{I}^{d_1 \times \ldots \times d_n}$. The *unrolled* notation represent a specific element $x_{i_1, i_2, \ldots, i_n}$ of the tensor X as X_k where the index k is computed using the row-major order, that is:

$$k = \sum_{l=1}^{n} (\prod_{p=1}^{l-1} d_p) i_l \tag{3}$$

4 CoCoNet

CoCoNet is a graphical tool conceived for aiding researchers in the design, conversion and property specification for building neural networks verification benchmarks. The graphical environment allows to select the building layers and arrange them accordingly for creating and saving a neural network in the *ONNX* file format, as well as to import a *PyTorch* file in order to convert it. It is also possible to define or import VNN-LIB specifications on the input and the output with dedicated interfaces.

4.1 Software Architecture

In Figure 1 we show the design abstractions of CoCoNet as an UML class diagram. The purpose of this diagram is not to provide a complete description of the software architecture, but to describe its fundamental structure.

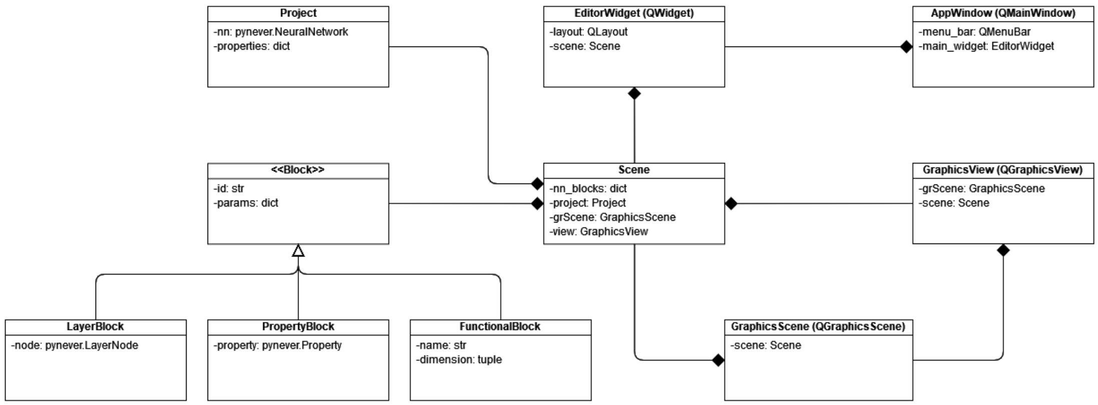


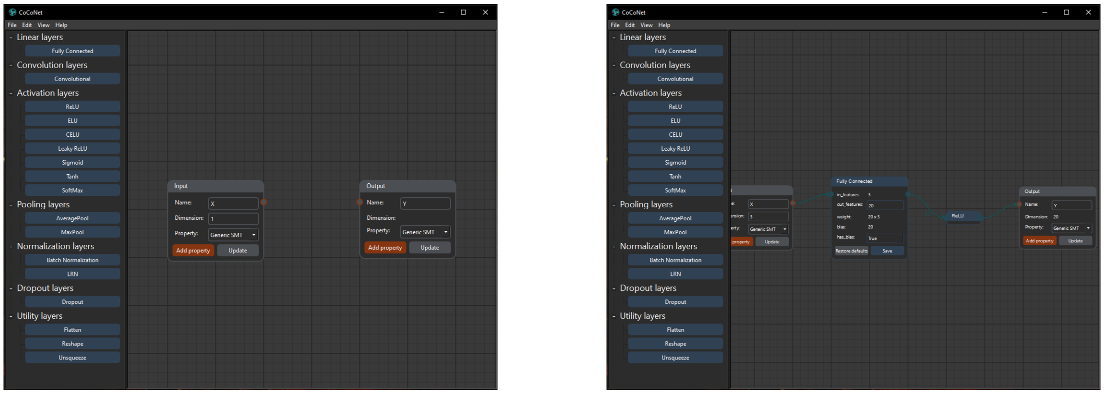
Figure 1: UML Class Diagram of CoCoNet with the main classes that describe the core design. Aggregation relationships are depicted with a solid diamond, inheritance relationships with a blank arrow.

The internal representation of a neural network is provided by pyNeVer [12], which is a Python API providing also learning and verification capabilities for NNs. Since CoCoNet is focused only on representation and conversion, here we will not cover the other functionalities that pyNeVer provides. The model representation is contained in a `Project` class, which groups all the functionalities required to manage the design of a network: it is the class that manages the interaction with pyNeVer and the procedures to open and save a neural network file, with or without the related properties.

In order to create a graphical interface we leveraged PyQt's [25] Graphics View framework which provides a surface for managing and interacting with a large number of custom-made 2D graphical items, and a view widget for visualizing the items, with support for zooming and rotation. The framework includes an event propagation architecture that allows precise interaction capabilities for the items on the scene. Graphics View uses a BSP (Binary Space Partitioning) tree to provide very fast item discovery, and as a result of this, it can visualize large scenes in real-time, even with millions of items. The framework builds on the interaction between the `Scene` class, which is a container of graphical items, and the `GraphicsView` class, which observes the Scene and renders a subset of the objects in a viewport. Here we provide a division between a `GraphicsScene` and the `Scene` by using `GraphicsScene` as a reference for objects to be created or destroyed as well as to set global parameters like the background pattern, and the `Scene` as the container of every object that is used in the application.

Finally, the elements displayed in the scene are concrete instances of the abstract class `Block`: in particular, we distinguish the `LayerBlock` for representing a concrete layer in the network, the `FunctionalBlock` for defining the input and the output of the network and the `PropertyBlock` for representing the VNN-LIB properties. The `Block` classes are structured in a way that allows them to be connected with multiple inputs and outputs such that future extensions of CoCoNet could easily support other architectures than feed-forward neural networks like ResNets and recurrent neural networks.

Thanks to pyNeVer it is also possible to directly import a neural network model in the ONNX or *PyTorch* format and visualize it in order to add properties or convert it. Leveraging the design of pyNeVer, which provides a generic interface for the conversion of models from



(a) The starting window displays the input and output nodes for defining the input dimension and the labels.

(b) When layers are added to the network the output dimension in set automatically and the input block is locked.

Figure 2: Interface of CoCoNet at launch (left) and after defining two layers (right).

and to its internal representation, anyone can write its own conversion for other file formats in order to extend the capabilities of the tool and support more benchmarks.

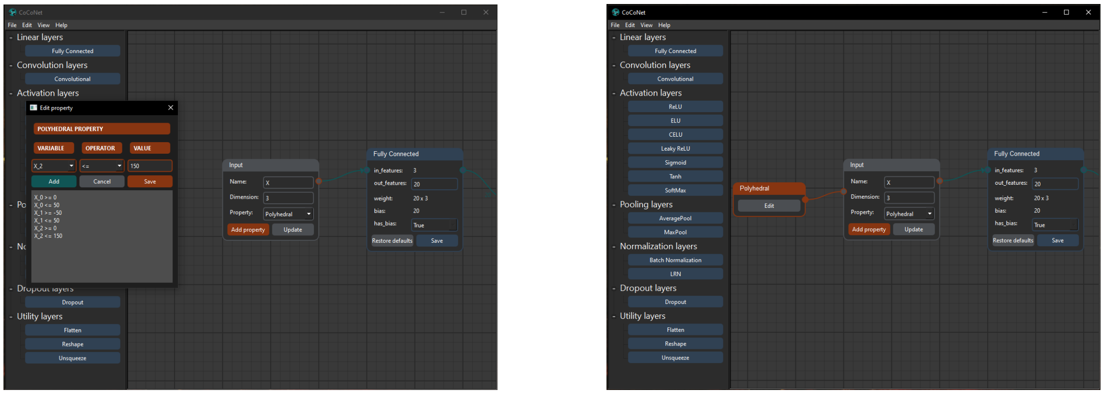
4.2 Demonstration

Now we show how to build from scratch a neural network for the example provided in Section 3.2 for the ACC application [8]. A network created within CoCoNet can be exported in *PyTorch* for training, and then re-imported in CoCoNet for exporting it in the VNN-LIB format.

Building the model. In Figure 2 we see side by side the starting screen of CoCoNet, where the two functional blocks are displayed in order to define the network input and the corresponding labels, and the definition of the first fully connected layer of 20 neurons, followed by a ReLU activation, which are by default automatically added sequentially to the network. For this reason we add each layer and update its parameters before adding another one, including the input block for specifying the input dimension. The values are updated clicking on the block's *Save* button, while *Restore defaults* resets the values to the default ones without overwriting.

Defining the property. Once the network is ready, it is possible to define VNN-LIB properties related to the input and the output. Following the description in [8], we want to bound the input variables following the *OutBounds* description: we can use the property selector in the input block to define a *Polyhedral* property, i.e., a controlled environment for bounding variables without needing to write plain SMT as in *Generic SMT* properties. Figure 3a shows the interface where the property is defined with the values described in Section 3.2 and Figure 3b shows the corresponding property block added to the view for editing or deleting the property.

Loading and saving models. CoCoNet supports direct loading of existing neural networks in the *ONNX* and *PyTorch* file formats and, provided it is defined on the same input and output identifiers, is able to open and link a property to a network, too. In order to produce



(a) The property editor allows to bound the selected variables in a controlled way, restricting their values.

(b) After defining a property, the corresponding block is displayed attached to the functional block.

Figure 3: Definition of a property in CoCoNet showing the interface for defining a polyhedral property (left) and the resulting block (right).

a benchmark for VNNCOMP one should be able to import the neural network model in CoCoNet as well as the property, then it is possible to create two files — a .onnx file for the model and a .smt2 file for the property — choosing the "Save as..." VNN-LIB entry in the menu.

Command-line interface. Finally, it is possible to leverage CoCoNet's capabilities using the command line with the options `-check <<model>>` or `-convert <<model>>`. The two options allow to check whether a *ONNX* model is compliant with the VNN-LIB standard and to convert a *PyTorch* model to the *ONNX* format, provided the operators are supported by the standard.

5 Conclusions

In this paper we presented our contribution to the verification of neural networks community in terms of VNN-LIB, a standard for the definition of verification benchmarks, and CoCoNet, a tool for the creation, visualization and conversion of benchmarks to the standard.

The purpose of our contribution is to provide researchers a stable ground for building benchmarks complying to the principles of VNNCOMP and to allow practitioners to experiment with a graphical user interface for visualizing neural networks and properties at once. We expect to integrate the standard with more operators and architectures in the future, and to keep CoCoNet aligned to such integrations so that it will always be possible to make use of it when preparing benchmarks, as well as to support more network formats such as Tensorflow, Keras and nnet. Overall, CoCoNet implements the features that can help contributors to check whether their models are compliant to the standard, and to convert already trained models in the *PyTorch* format to the ONNX format without the need of re-implementing them.

The current version of VNN-LIB and CoCoNet is intended to support basic sequential architectures, but the SMT-LIB language is already expressive enough to model complex properties, albeit with no temporal structure. An effort to deal with complex specifications for neural networks in the ONNX format is also proposed in [7], where it is described how to compile VNN-LIB specification starting from a high-level DSL. Our future objectives are to upgrade the VNN-LIB standard — and likewise CoCoNet — to be able to cope with more complex architectures such as non-sequential neural networks, i.e., ResNets, and Recurrent architectures. For the latter, an extension of the basic SMT-LIB language to deal also with temporal properties could be in order.

Acknowledgements

The design of CoCoNet's interface was part of a M.Sc. project carried by Andrea Gimelli and Giacomo Rosato, whose contribution is gratefully acknowledged.

References

[1] Junjie Bai, Fang Lu, and Ke Zhang. ONNX: Open Neural Network Exchange, https://github.com/onnx/onnx, 2023.

[2] Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings*, volume 12673 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2021.

[3] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[4] Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.

[5] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. First three years of the international verification of neural networks competition (vnn-comp). *arXiv preprint arXiv:2301.05815*, 2023.

[6] Christopher Brix and Thomas Noll. Debona: Decoupled boundary network analysis for tighter bounds and faster adversarial robustness proofs. *arXiv preprint arXiv:2006.09040*, 2020.

[7] Matthew L. Daggitt, Wen Kokke, Robert Atkey, Luca Arnaboldi, and Ekaterina Komendantskya. Vehicle: Interfacing neural network verifiers with interactive theorem provers. 2022.

[8] Stefano Demarchi, Dario Guidotti, Andrea Pitto, and Armando Tacchella. Formal verification of neural networks: a case study about adaptive cruise control. In *International ECMS Conference on Modeling and Simulation*, pages 310–316, 2022.

[9] Claudio Ferrari, Mark Niklas Müller, Nikola Jovanovic, and Martin T. Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[11] Dario Guidotti, Stefano Demarchi, Armando Tacchella, and Luca Pulina. The Verification of Neural Networks Library (VNN-LIB), www.vnnlib.org, 2023.

[12] Dario Guidotti, Luca Pulina, and Armando Tacchella. pyNeVer: A framework for learning and verification of neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 357–363. Springer, 2021.

[13] Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pages 2513–2520. IOS Press, 2020.

[14] Patrick Henriksen and Alessio Lomuscio. Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In *IJCAI*, pages 2549–2555, 2021.

[15] Holger H Hoos and Thomas Stützle. Satlib: An online resource for research on sat. *Sat*, 2000:283–292, 2000.

[16] Hadi Jahanbakhti, Mahdi Pourgholi, and Alireza Yazdizadeh. Online neural network-based model reduction and switching fuzzy control of a nonlinear large-scale fractional-order system. *Soft Computing*, March 2023.

[17] 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. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, pages 443–452, 2019.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[19] Paolo Marin, Massimo Narizzano, Luca Pulina, Armando Tacchella, and Enrico Giunchiglia. Twelve years of qbf evaluations: Qsat is pspace-hard and it shows. *Fundamenta Informaticae*, 149(1-2):133–158, 2016.

[20] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results. *arXiv preprint arXiv:2212.10376*, 2022.

[21] Vineel Nagisetty. Domain knowledge guided testing and training of neural networks. Master's thesis, University of Waterloo, 2021.

[22] A. Pavithra, G. Kalpana, and T. Vigneswaran. Deep learning-based automated disease detection and classification model for precision agriculture. *Soft Computing*, March 2023.

[23] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 243–257, 2010.

[24] Luca Pulina and Armando Tacchella. Challenging SMT solvers to verify neural networks. *AI Commun.*, 25(2):117–135, 2012.

[25] PyQT. Pyqt reference guide. 2012.

[26] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. Boosting robustness certification of neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[27] Geoff Sutcliffe, Christian Suttner, and Theodor Yemenis. The tptp problem library. In *Automated Deduction—CADE-12: 12th International Conference on Automated Deduction Nancy, France, June 26–July 1, 1994 Proceedings 12*, pages 252–266. Springer, 1994.

[28] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[29] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1701–1708, 2014.

[30] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[31] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I*, pages 3–17. Springer, 2020.

[32] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 34, 2021.

[33] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

[34] Dong Yu, Geoffrey E. Hinton, Nelson Morgan, Jen-Tzung Chien, and Shigeki Sagayama. Introduction to the special section on deep learning for speech and language processing. *IEEE Trans. Audio, Speech & Language Processing*, 20(1):4–6, 2012.

[35] Baowen Zhang, Wei Huang, and Fengnian Zhao. An available-flow neural network for solving the dynamic groundwater network maximum flow problem. *Soft Computing*, March 2023.

[36] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in Neural Information Processing Systems*, 2022.

[37] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31:4939–4948, 2018.