Case Study: Neural Network Malware Detection Verification for Feature and Image Datasets

Preston K. Robinette
preston.k.robinette@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Diego Manzanas Lopez
diego.manzanas.lopez@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Serena Serbinowska
serena.serbinowska@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Kevin Leach
kevin.leach@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Taylor T. Johnson
taylor.johnson@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

ABSTRACT

Malware, or software designed with harmful intent, is an ever-evolving threat that can have drastic effects on both individuals and institutions. Neural network malware classification systems are key tools for combating these threats but are vulnerable to adversarial machine learning attacks. These attacks perturb input data to cause misclassification, bypassing protective systems. Existing defenses often rely on enhancing the training process, thereby increasing the model's robustness to these perturbations, which is quantified using verification. While training improvements are necessary, we propose focusing on the verification process used to evaluate improvements to training. As such, we present a case study that evaluates a novel verification domain that will help to ensure tangible safeguards against adversaries and provide a more reliable means of evaluating the robustness and effectiveness of anti-malware systems. To do so, we describe malware classification and two types of common malware datasets (feature and image datasets), demonstrate the certified robustness accuracy of malware classifiers using the Neural Network Verification (NNV) and Neural Network Enumeration (nnenum) tools[1], and outline the challenges and future considerations necessary for the improvement and refinement of the verification of malware classification. By evaluating this novel domain as a case study, we hope to increase its visibility, encourage further research and scrutiny, and ultimately enhance the resilience of digital systems against malicious attacks.

1 INTRODUCTION

Malware is software intentionally designed to cause damage, gain control, or disrupt an interface, network, or digital device. There are many use cases for malware, including stealing sensitive data (data theft), stealing data and holding it for ransom (ransomware attacks), the creation of a 'botnet' (Distributed Denial of Service (DDoS)), using someone else's system to mine cryptocurrency (cryptojacking), monitoring activity and passwords (espionage), and distributing spam. Not only do malware attacks pose a significant threat to individuals, but they also frequently target a wide array of institutions. These include educational institutions like universities, businesses both private and public, healthcare entities, and governmental bodies. In a 2022 Response Report from Palo Alto Networks, reported ransomware payouts in the US reached up to $8 million USD with demands received up to $30 million USD as a result of ransomware attacks [1]. Malware represents a significant and ever-evolving threat to digital security, privacy, and integrity, and as such, is imperative to protect against.

A key tool against this digital threat is the use of malware classification systems, which classify incoming and existing software. Trained models can detect malware in real-time, preventing systems from running the classified software, even if a user interacts with it. Most malware classification systems today utilize machine learning, and while adept at classification, they are also vulnerable to attacks. Bad actors can bypass the use of malware classification systems by utilizing adversarial machine learning. Adversarial machine learning relies on small perturbations in the input space to cause misclassification in the target system, as shown in Figure 1. While adversarial attacks are applicable to all input types, this example demonstrates an adversarial attack on an image representation of a malware binary, which is a common classification technique in the malware domain. In this example, prior to the adversarial attack, the correct class "Yuner.A" is correctly predicted. This original malware image is then added with a small percentage ϵ of a noisy sample a from a different class, resulting in the misclassification shown on the far right of the image ($x + \epsilon(a)$).Adversarial attacks are not only imperceptible to the human eye, but they are also difficult to detect via anomaly detection because they are close to the original image. In [5], the authors show that anomaly detection defenses such as variational autoencoders and generative adversarial networks are not robust to adversarial examples. Most defense mechanisms, therefore, rely on improvements in the training process, such as distillation [24], adversarial retraining [20], randomized smoothing [4, 8, 17, 38], and randomized deletion [11]. Each of these methods increases the robustness of a trained model to perturbations in the input space, which can be evaluated quantitatively as the

[1]Code, VNN-LIB: https://github.com/pkrobinette/verify_malware

FormaliSE '24, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0589-2/24/04
https://doi.org/10.1145/3644033.3644372

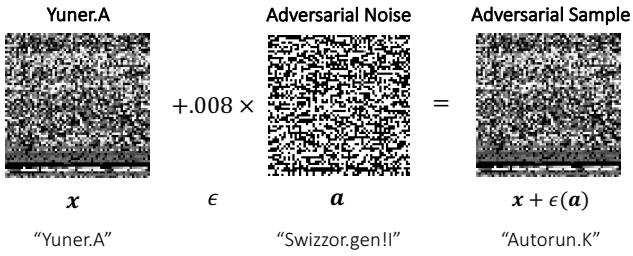


Figure 1: Adversarial machine learning example on Malimg dataset. Prior to the adversarial attack, the correct class "Yuner.A" is correctly predicted. This original malware image is then added with a small percentage ϵ of a noisy sample a from a different class, resulting in the misclassification shown on the far right of the image ($x + \epsilon(a)$).

percentage of test inputs to be correctly classified within a bounded distance of a starting sample [6].

While the robustness of trained models has been used to verify classification networks in previous works, the focus has been on implementation improvements rather than on the domain itself. For instance, using robustness as a metric to show the improvements of using a robust training method as demonstrated on an image labeling dataset like CIFAR-10 [7]. Just as implementations improve under scrutiny, so too does a domain benefit from rigorous analysis, increased visibility, and continuous refinement, which is especially important for a domain as safety-critical as malware classification.

Toward this end, we present an initial case study on malware classification verification and put forward the concept of 'meaning' to a perturbation. For instance, we acknowledge the possibility of applying an $\mathcal{L}_\infty$ perturbation on a binary feature; however, this does not yield any significant or meaningful semantic implications to the original binary file. These efforts work towards providing tangible guarantees against bad actors seeking to bypass classification schemes and provide valuable insight into a model's reliability to protect safety-critical networks. To do so, we describe two common malware dataset types and demonstrate the certified robustness accuracy of two types of malware classifications using the Neural Network Verification (NNV) [19, 33] and Neural Network Enumeration (nnenum) [2] tools. Through this process, we identify domain challenges and areas to be considered moving forward. The contributions of this work are the following:

(1) Novel application of formal verification to neural network malware classification (binary and family).
(2) Identify 'meaningful' perturbations for two different types of malware datasets (feature and image) and classification types (binary and family).
(3) Outline the challenges and future considerations necessary for the improvement and refinement of the verification of neural network malware classification.

2 PRELIMINARIES

In this section, we introduce the malware datasets used in this work, the types of malware classification to which the datasets are applied, NNV and nnenum tools for certifying the robustness of trained models, and the metrics used to evaluate trained models.

2.1 Malware Datasets

Below, we discuss two commonly-used types of malware datasets: (1) feature datasets, which represent malware samples as vectors of data such as byte entropy and string length, and (2) image datasets, which represent malware samples as grayscale images.

2.1.1 Feature Datasets. Malware feature datasets are typically composed of characteristics or *features* extracted from a variety of malware samples. The features can include static attributes such as file size, header information, and/or the presence of specific strings, as well as dynamic attributes that are revealed when the malware is executed in a controlled environment, such as API calls, network activity, and/or changes to the file system or registry. The extraction of these features can be conducted through a combination of static and dynamic analysis. Static analysis involves examining the malware code without executing it, and dynamic analysis involves running the malware in a sandboxed environment and observing and recording its behavior.

BODMAS The feature dataset used in this work is the Blue Hexagon Open Dataset for Malware Analysis (BODMAS), which is a collection of timestamped malware and benign samples for research purposes, co-created with Blue Hexagon [39]. It includes 57,293 malware (label 0) and 77,142 benign samples (label 1) collected between August 2019 and September 2020. The LIEF project was used to extract 2381 feature vectors from each sample using dynamic and static analysis, along with its classification label as benign or malicious. The extracted features can be broken down into seven different types, as shown in Table 1. While the data can also be described by feature groups, we focus on data types as they influence the chosen specifications used for verification.

While features extracted with dynamic and static analysis offer valuable insights into a sample, they each have their own set of challenges. Static analysis can struggle against obfuscation techniques such as code encryption, packing, and the use of non-standard code constructs, which are used by malware authors to hide their code's true intent. Additionally, static analysis can be time-consuming and resource-intensive when dealing with large and complex pieces of software and result in high false positive rates. Some sophisticated malware can even detect when they are being executed in a sandbox or virtual environment, altering their behavior or refusing to run altogether, thereby evading detection.

2.1.2 Image Datasets. Due to the difficulties of creating and maintaining up-to-date malware datasets using feature extraction, researchers also use image representations of software for training data [14, 16, 23, 27, 35]. This method works by converting binary executable files of malware into grayscale images, with each pixel in the image corresponding to a byte in the binary file, as shown in Figure 2. These images can then be analyzed using image processing techniques or deep learning models originally designed for image recognition tasks, such as convolutional neural networks (CNNs). The process begins with the raw bytes of the malware binary being aggregated into an 8-bit vector. Each of these 8-bit vectors is then converted into a decimal value to fit the grayscale range (0-255).

Table 1: BODMAS dataset feature types and examples. The ranges of values for each feature type are drastically different, showing the importance of using range-specific perturbations during verification.

Feature Type	Count	Max Range Pre-Scale	Max Range Post-Scale	Example
Continuous	5	[5.0, 2.0e5]	[-0.1, 304.6]	Entropy of the file
Categorical	8	[0.0, 6.5e4]	[-0.0, 124.3]	Machine type
Hash Categorical	560	[-647.6, 15.4]	[0.0, 361.0]	Hash of original file
Discrete with Large Range	34	[0.0, 4.3e9]	[-0.0, 261.6]	Number of occurrences of each byte value within the file
Binary	5	[0.0, 1.0]	[-2.1, 0.5]	Presence of debug section
Hash Categorical Discrete	1531	[-8.0e6, 1.6e9]	[-327.9, 164.0]	Hash of target system type
Memory Related	16	[0.0, 4.0e9]	[-0.1, 307.5]	Size of the original file
Null	222	[-31.0, 60.0]	[-0.9, 160.4]	—

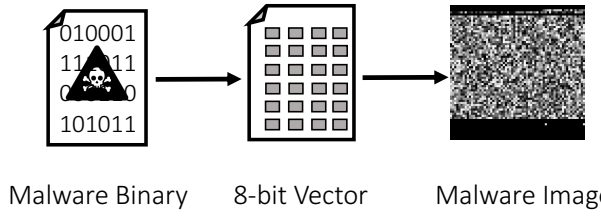


Figure 2: Malware binary conversion to an image. The malware binary is preprocessed to extract sequences of 8-bit vectors, or bytes. Each byte is then mapped to a pixel in the image, which is then plotted on a grid to create an image.

These values are then reshaped into a two-dimensional array, effectively creating an image. The resulting image, often referred to as a *byteplot*, can reveal patterns that might be indicative of certain types of malware, as shown in Figure 3.

Malimg The malware image dataset used in this work is the Malimg Dataset, which is composed of 9339 malware images from 25 different malware families [23]. The malware families are shown in Table 7.

2.2 Malware Classification Types

A malware classifier is an automated way to categorize and differentiate software (benign vs. malicious). In addition to the canonical classification of malicious vs. benign, malware classifiers are also used to identify a known malware family. A malware family refers to a group of malware that share significant characteristics and are often developed from the same code base. These shared characteristics can include similar behavior patterns, payload delivery methods, or purpose. By identifying a sample's malware family, cybersecurity professionals gain insight into the nature, origin, purpose of the sample, threat level and type, propagation mechanisms, and potential damage. Additionally, successful identification can often lead to the identification of the creator's signature, which can help to identify the attack group responsible. All of these pieces of information aid in the subsequent development of effective countermeasures to neutralize or mitigate the effects of the detected malware. In this work, we train both **binary** classifiers ($Y_{\text{binary}} = \{\text{malicious}, \text{benign}\}$) and **family** classifiers ($Y_{\text{family}} = \{\text{Adialer.C}, \text{Agent.FYI}, \text{Allaple.A}, \ldots, \text{Yuner.A}\}$), where Y is the set of all output classes.

2.3 Neural Network Verification

To analyze the neural network malware classifiers, we make use of NNV and nnenum tools. NNV is a software tool written in MATLAB[2] that implements reachability methods to formally verify specifications of neural networks (NN) [19, 33]. NNV uses a star-set state-space representation and reachability algorithms that allow for a layer-by-layer computation of exact or overapproximate reachable sets for feed-forward (FFNN), convolutional (CNN) [29], Semantic Segmentation (SSNN) [32], and recurrent (RNN) [31] neural networks, as well as neural ordinary differential equations [21], and neural network control systems (NNCS) [18].

nnenum is a software tool written in Python that addresses the verification of ReLU neural networks through optimized abstraction refinement [2]. This approach combines zonotopes with star set overapproximations for feed-forward (FFNN), convolutional (CNN), and recurrent (RNN) neural networks. In this work, we focus on one type of property to verify: robustness, which is described in Definition 2.1.

Definition 2.1 (Robustness). Given a neural network classifier f, input $x \in \mathbb{R}^{n \times m}$, target $y \in \mathbb{R}^N$ where N is the number of classes (i.e., $N = 2$ for binary classification, $N = 25$ for family classification), perturbation parameter $\epsilon \in \mathbb{R}$, and an input set R containing x_p such that $X_p = \{x : ||x - x_p|| \leq \epsilon\}$ which represents the set of all possible perturbations of x where $||x - x_p||$ is the $\mathcal{L}_\infty$ norm. The classifier is locally **robust** at x if it classifies all the perturbed inputs x_p to the same label as y, i.e., the system is **robust** if $f(x_p) = f(x) = y$ for all $x_p \in X_p$.

2.4 Metrics

In this section, we introduce the classifier and verification performance metrics used in this work.

Classifier Performance. To evaluate trained models prior to verification, we utilize accuracy, precision, recall, and F1 score

[2]MATLAB [2022b]: https://www.mathworks.com/products/new_products/r2022b-transition.html

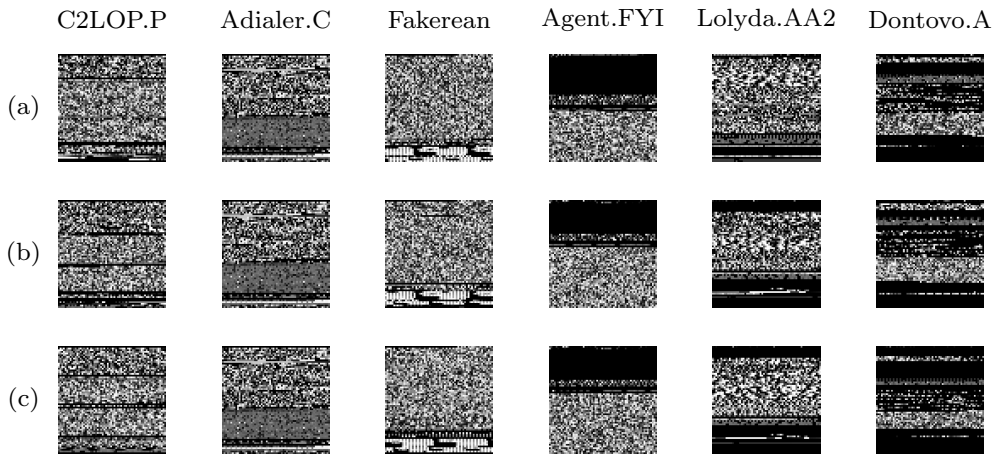


Figure 3: Malware family image examples. Each of the malware families produce relatively distinct patterns, making image classification an effective tool for malware detection and categorization. (a), (b), and (c) correspond to different samples of the same family indicated by the column name.

metrics. We formalize these metrics across classes Y using *True Positives* (TP_{y_i}): samples **correctly** classified as class y_i, *True Negatives* (TN_{y_i}): samples **correctly** classified as **not** in class y_i, *False Positives* (FP_{y_i}): samples **incorrectly** classified as class y_i, and *False Negatives* (FN_{y_i}): samples **incorrectly** classified as **not** in class y_i, where $y_i \in Y$ s.t. $Y = \{y_1, y_2, ..., y_N\}$ and N is the number of all classes in the dataset. The equations for accuracy, precision, recall, and F1 score are shown in Equations 1, 2, 3, and 4 respectively.

$$\text{Accuracy} = \frac{\sum_{i=1}^{Y} \text{TP}_{y_i} + \text{TN}_{y_i}}{\text{Total Samples}} \tag{1}$$

$$\text{Precision} = \frac{1}{N} \sum_{i=1}^{Y} \frac{\text{TP}_{y_i}}{\text{TP}_{y_i} + \text{FP}_{y_i}} \tag{2}$$

$$\text{Recall} = \frac{1}{N} \sum_{i=1}^{Y} \frac{\text{TP}_{y_i}}{\text{TP}_{y_i} + \text{FN}_{y_i}} \tag{3}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Verification Performance. The verification strategy implemented in NNV to certify the robustness of the BODMAS and Malimg benchmarks is described in Algorithm 2.1. This algorithm consists of two main steps and three potential outcomes:

(1) **Falsification** (0): In the first step of the verification process, we classify 500 random examples from the created input set (based on bound from adversarial attack). If a sample is misclassified, or a counterexample is found, this is considered a successful falsification, and the model is not robust to the perturbation ϵ.
(2) **Robust** (1): If no counterexamples are found, then we proceed to run the reachability analysis: starting with an over-approximation method referred to as *relax-approx-area* with a relax factor of 0.5. If the input set with *relax-approx-area* evaluates to the correct class, the model is considered robust, and the valuation returns a 1.
(3) **Unknown** (2): If the verification result from *relax-approx-area* is unknown, we proceed to run the *approx-star* method, which is less conservative than the relax-star, but also more computationally expensive. If the sample with *approx-star* is found to be robust, the valuation is 1; otherwise, the valuation is 2, which is considered unknown, as further refinement might or might not prove the model to be robust to perturbation ϵ.

Algorithm 2.1 Verification strategy

Input: $f, x, target, \mathcal{A}, N_R$ ▷ NN, input, label target, attack, # of random examples
Output: res ▷ verification result

```
procedure res = malware_verification(f, x, A, N_R)
    X_rand = genRandExamples(x, A, N_R) ▷ generate random examples based on attack
    for i = 1 : N do
        y = f(X_rand(i)) ▷ classify random image
        if y ≠ target then
            res = 0 ▷ counterexample found
            return res ▷ stop procedure, no reachability analysis needed
    I = constructInputSet(x, A) ▷ construct input set
    R = Reach(f, I, relax) ▷ compute reachable set using relax-star-area reachability
    res = verify(R, target) ▷ verify output set against target class
    if res ≠ 1 then ▷ if result is unknown, try again with approx
        R = Reach(f, I, approx) ▷ compute reachable set using approx-star reachability
        res = verify(R, target) ▷ verify output set against target class
    return res ▷ return verification result {0 = not robust, 1 = robust, 2 = unknown}
```

In addition to this verification step, we also produce VNN-LIB files [9] for each dataset, sample, and epsilon value to be used by the nnenum verification tool. VNN-LIB is a standard file type to represent the specification of neural networks based on input-output

Table 2: Neural network classifier model names and architectures used to train binary and family classifiers.

Classifier Type	Model Name	Model Architecture (in → [hidden layers] → out)
BODMAS (Binary)	none-2	2381 → 2
	4-2	2381 → [4] → 2
	16-2	2381 → [16] → 2
Malimg (Family)	linear-25	4096 → 25
	4-25	4096 → [4] → 25
	16-25	4096 → [16] → 25

relationships, derived from the Satisfiability Modulo Theories Library (STM-LIB)[3]. A VNN-LIB file consists of input bounds and a specification to verify. For instance, a VNN-LIB file for a Malimg dataset sample (64x64 image) with $\epsilon = 2/255$ would consist of 4096 input variables (the pixels of the image) with a range $\pm 2/255$ of the original pixel value. The specification would then be that the true label is maintained (robust) for this range of inputs. As we need to create a different VNN-LIB file for each sample and epsilon value, there are 1200 VNN-LIB files for the BODMAS dataset (100 samples * 4 feature types * 3 epsilon values) and 375 VNN-LIB files for the Malimg dataset (125 samples * 3 epsilon values). A single VNN-LIB file and a corresponding trained model in an Open Neural Network Exchange (ONNX)[4] format are then used as inputs to the nnenum tool, which presents three potential outcomes: Robust (*holds*), Falsification (*violated*), and Unknown (*timeout*).

During verification, we evaluate models using two metrics: (1) average time (s) to verify each input in the verification set for a given perturbation and (2) the number of inputs from the verification set for a perturbation (ϵ) that are certified robust according to Definition 2.1 over the number of total samples, i.e., certified robustness accuracy (CRA). For each model that is tested, the verification set consists of randomly selected samples from the dataset. For instance, models from the BODMAS dataset are evaluated on 100 inputs. These 100 inputs are then used for verification in subsequent rounds, where each round corresponds to a particular perturbation size ϵ. The average time metric would then be the average wall time it takes to verify across all 100 samples, and the CRA metric would be the number of these 100 samples that evaluate to Robust (1 for NNV, *holds* for nnenum) over the number of total samples.

3 VERIFICATION OF MALWARE CLASSIFIERS OVERVIEW

We aim to verify two different types of malware classifiers: binary classification (benign vs. malicious) and family classification (malware family). Each experiment contains three different phases: (1) training, (2) testing, and (3) verification.

[3]SMT-LIB: https://smtlib.cs.uiowa.edu
[4]ONNX: https://onnx.ai/

Table 3: Perturbations (ϵ) used during verification.

Classifier Type	Coverage	ϵ		
BODMAS	All	0.01%	0.05%	0.1%
	Discrete & Continuous	0.01%	0.05%	0.1%
	Discrete	0.1%	0.5%	1%
	Continuous	1%	5%	10%
Malimg	All	$\frac{1}{255}$	$\frac{2}{255}$	$\frac{3}{255}$

3.1 Binary Classification

To test the verification of binary malware classifiers (benign vs. malicious), we use the BODMAS dataset described in Section 2.1.1. Prior to training, the data is scaled using $z = \frac{x-\mu}{\sigma}$, where z is the standardized score, x is a given sample in a feature column, μ is the mean of the feature, and σ is the standard deviation of the feature. The standard scalar standardization is used due to the large differences in value range between potential features.

Each of the remaining experimental phases is described in more detail below:

(1) **Training:** Three different binary classifiers with varying architectures are trained using this data. The architectures used in these experiments are shown in Table 2. If a hidden layer is present, a ReLU activation function is used on the outputs of the hidden layer. These architectures are trained with sparse categorical cross-entropy loss function, Adam optimizer, 40 epochs, and a batch size of 128 samples.
(2) **Testing:** To create a baseline for verification performance, we then test each classifier on the test data. From this, we gather accuracy, F1, precision, and recall metrics described in Section 2.4.
(3) **Verification:** We then use NNV and nnenum with the epsilon values shown in Table 3 to obtain the number verified and time to verify metrics for 100 randomly selected test samples. The randomly selected test samples are the same across all verification experiments. The randomly selected samples reflect the distribution of the data, which consists of 43% malicious samples. Additionally, as the BODMAS dataset consists of different types of features and ranges, the perturbation applied to each feature is the epsilon value percentage of the range for that particular feature. For instance, if feature 1 has a range $[3, 567]$, the $\mathcal{L}_\infty$ bound with $\epsilon = 0.1\%$ would be $\pm 0.56 = 0.1\% \times (567 - 3)$. This provides a more feature realistic perturbation to each sample.
   (a) **All**: As a baseline, the epsilon perturbation (a) is applied to all 2381 features in a sample. While applying the epsilon perturbation to all features gives insight into the complete robustness of a model, a perturbation is not meaningful or impactful across all features. For instance, applying an $\epsilon = 0.1\%$ perturbation to a binary feature. While not all perturbed samples will be semantically meaningful, the $\mathcal{L}_\infty$ bound will contain some samples that are, providing a high-level evaluation of the model.

Table 4: Training results for BODMAS (binary) models and Malimg (family) models. Each of the models performs well according to accuracy, precision, recall, and F1 score metrics.

Dataset	Model	Accuracy	Precision	Recall	F1
BODMAS	none-2	0.99	0.98	0.99	0.99
	4-2	0.99	0.99	0.99	0.99
	16-2	0.99	0.99	0.99	0.99
Malimg	linear-2	0.99	0.98	0.97	0.97
	4-2	0.98	0.97	0.96	0.97
	16-2	0.99	0.97	0.96	0.97

Table 5: Certified robustness accuracy (CRA) and average time to verify verification results on 100 samples from the BODMAS (binary) dataset using NNV and nnenum verification tools on various feature types and ($\mathcal{L}_\infty$) perturbations.

Metric	Model	Tool	All			Continuous & Discrete			Discrete			Continuous		
			0.01 %	0.05 %	0.1 %	0.01 %	0.05 %	0.1 %	0.1 %	0.5 %	1 %	1 %	5 %	10 %
CRA (%)	none-2	NNV	94	61	19	98	98	96	96	90	68	95	84	56
		nnenum	94	61	19	98	98	96	96	90	68	95	84	56
	4-2	NNV	99	69	26	100	100	99	100	98	88	100	97	81
		nnenum	99	73	31	100	100	99	100	98	90	100	97	81
	16-2	NNV	100	65	27	100	100	100	100	99	91	100	96	82
		nnenum	100	71	34	100	100	100	100	99	92	100	96	82
Avg. Time (s)	none-2	NNV	0.35	0.30	0.29	0.17	0.17	0.17	0.27	0.27	0.28	0.31	0.26	0.19
		nnenum	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
	4-2	NNV	0.31	0.42	0.62	0.17	0.17	0.17	0.28	0.28	0.31	0.29	0.27	0.28
		nnenum	0.50	0.50	0.51	0.50	0.50	0.51	0.50	0.50	0.51	0.50	0.50	0.51
	16-2	NNV	0.28	0.64	1.04	0.18	0.18	0.18	0.26	0.28	0.33	0.28	0.31	0.27
		nnenum	0.50	0.50	0.52	0.50	0.50	0.52	0.50	0.50	0.52	0.50	0.50	0.52

(b) **Discrete & Continuous**: From the available feature types, discrete and continuous features are the most applicable to $\mathcal{L}_\infty$ perturbations. As such, we apply epsilon perturbations (b) to only continuous and discrete features.
(c) **Discrete**: Discrete features offer insight into details like the number of imported libraries, the number of exported libraries, and the number of symbols in the file. As most of the discrete features involve counts of elements in the sample, small value changes can have a drastic impact on the classification of a sample. The epsilon values (c) chosen reflect this property.
(d) **Continuous**: In contrast to the discrete features, the continuous features generally result from calculations based on the elements in the sample, such as the entropy of strings or the average length of strings. These elements are more robust to perturbations, and larger epsilon values (d) were selected to test these elements.

3.2 Family Classification

To test the verification of malware family classifiers, we utilize the Malimg dataset described in Section 2.1.2, which consists of image representations of malware samples from 25 different families. Prior to training, each of the images is resized to size 64x64, and the pixel values are normalized between 0 and 1. The data is then split into two different sets: training (8404 images) and testing (935 images).

(1) **Training:** Three different family classifiers with varying architectures are trained using this data. The architectures used in these experiments are shown in the bottom part of Table 2. If hidden layers are present, they represent a convolutional layer. Additionally, these convolutional models utilize a ReLU activation function. These architectures are trained with a categorical cross-entropy loss function, Adam optimizer, 10 epochs, and a batch size of 64 samples.
(2) **Testing:** To create a baseline for verification performance, we then test each classifier on the test data. From this, we gather accuracy, F1, precision, and recall metrics.
(3) **Verification:** We then use NNV and nnenum with the epsilon values shown in Table 3 to obtain the number verified and time to verify metrics for 125 samples. To create this set, 5 samples from each class in the validation set are randomly selected. Each verification experiment utilizes the same 125 images. Out of the 25 classes in the validation data, 20 of them contain 25 images, giving the verification set 20% coverage on most classes. As the Malimg dataset consists of images, these epsilon values are applied to pixels in the sample image.

Table 6: Certified robustness accuracy (CRA) and average time to train verification results on 5 samples from each class (125 samples) from the Malimg (family) dataset using NNV and nnenum verification tools on pixel level ($\mathcal{L}_\infty$) perturbations.

Metric	Model	Tool	Epsilon (ϵ) 1/255	2/255	3/255
CRA (%)	linear-25	NNV	85	83	79
		nnenum	90	86	82
	4-25	NNV	89	76	62
		nnenum	94	80	66
	16-25	NNV	88	82	67
		nnenum	90	86	64
Avg. Time (s)	linear-25	NNV	0.84	0.85	0.85
		nnenum	3.60	3.63	3.69
	4-25	NNV	17.75	41.66	82.18
		nnenum	11.59	10.80	11.13
	16-25	NNV	85.00	210.00	710.25
		nnenum	38.66	44.16	43.43

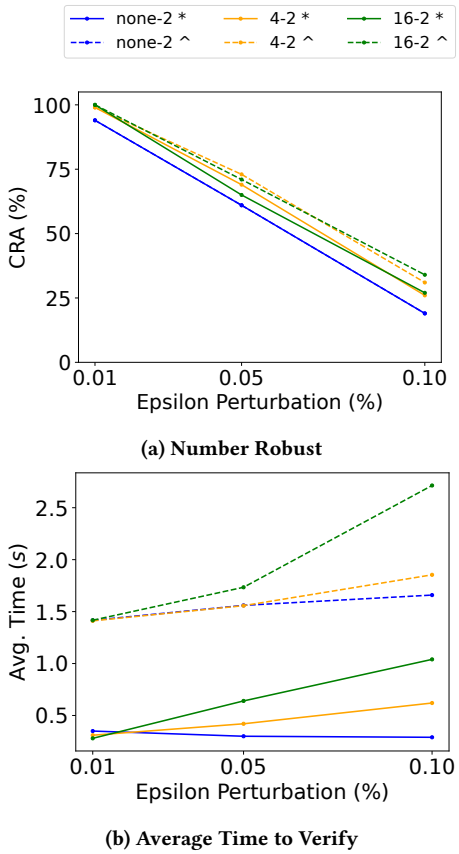
4 RESULTS

In this section, we present the results of our evaluation. These experiments were conducted on a macOS with a 2.3 GHz 8-Core Intel Core i9 processor with 16 GB 2667 MHz DDR4 of memory.

4.1 BODMAS

4.1.1 Model Performance. Each of the binary classification models trained on the BODMAS feature dataset achieves high-performance metrics, as indicated by Table 4. From these metrics, the trained models can accurately distinguish between benign and malicious samples. The high precision value indicates that when a model predicts an instance as malicious (malware), it is likely correct, and the high recall value means that a malware sample is rarely misclassified as benign. While providing valuable immediate feedback on the malware classification abilities, these metrics do not provide insight into the classifier's ability against adversarial examples.

4.1.2 Verification Performance. **(All)** The certified robustness accuracy (CRA) and average time to verify for the NNV and nnenum verification tools on *all* BODMAS features are shown in Figure 4. As the perturbation ϵ increases, the CRA of each of the models (*none-2, 4-2, 16-2*) decreases with both NNV and nnenum. Regarding model size, the results indicate that a larger model is more robust to a $\mathcal{L}_\infty$ perturbation, which is increasingly important as the perturbation size increases. While a larger model is more robust, it takes longer to verify, as shown by Figure 4b. This makes sense, as a larger perturbation would result in a larger created input set. While not affecting the falsification step, this will affect both the *relax-approx-area* method and *approx-star* method, which is slower for larger models. While the verification tool (NNV vs. nnenum) has little impact on the robustness results, NNV takes less time to verify for each of the perturbation sizes and models.



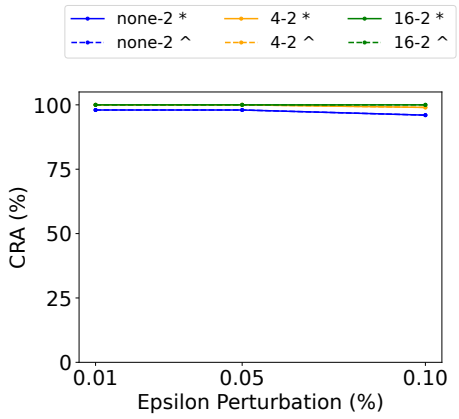
(a) Number Robust

(b) Average Time to Verify

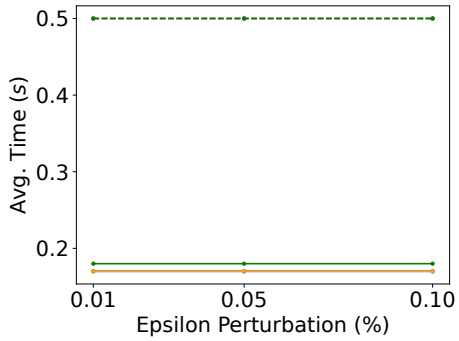
Figure 4: BODMAS (binary) robustness results ($\mathcal{L}_\infty$) for *all* features using NNV (*) and nnenum (∧) verification tools. As the perturbation size increases, CRA decreases and time to verify increases. A larger model is more robust to perturbations, especially as the perturbation size increases. While NNV and nnenum have similar robustness accuracy results, NNV takes less time to verify than nnenum for *all* features.

(Discrete & Continuous) The results of only perturbing *discrete & continuous* features are shown in Figure 5. Whereas model size impacts CRA for perturbations applied to *all* features, when applied to *discrete & continuous* features, model size does not heavily impact CRA, as seen by the close lines in Figure 5a. Additionally, the time to verify is not significantly impacted by model size either. This experiment highlights the importance of using meaningful features. While the epsilon perturbation sizes for *all* and *discrete & continuous* are the same (0.01%, 0.05%, 0.10%), the CRA results are drastically different. For $\epsilon = 0.1\%$, 19% of samples are verified robust for *all* features and 96% of samples are verified robust for *continuous & discrete* features. The feature type, therefore, impacts the robustness verification process. Regarding the verification tools, NNV and nnenum reach similar CRA results, but NNV is faster than nnenum for each model and epsilon perturbation size, as demonstrated by the results in Table 5.

(Discrete) The results from applying perturbations to just *discrete* features are shown in Figure 6. Whereas model size does not

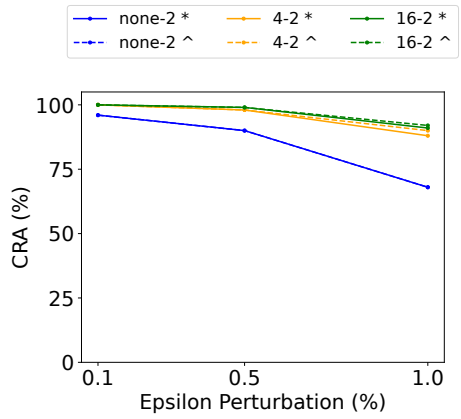


(a) Number Robust

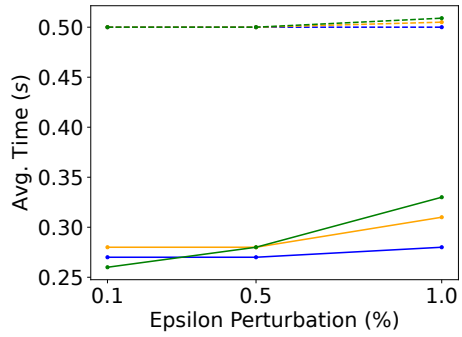


(b) Average Time to Verify

Figure 5: BODMAS (binary) robustness results ($\mathcal{L}_\infty$) for *discrete & continuous* features using NNV (*) and nnenum (∧) verification tools. Model size does not significantly affect CRA or time to verify for these features. While NNV and nnenum have similar robustness results, NNV takes less time to verify than nnenum for each model and perturbation size.



(a) Number Robust



(b) Average Time to Verify

Figure 6: BODMAS (binary) robustness results ($\mathcal{L}_\infty$) for *discrete* features using NNV (*) and nnenum (∧) verification tools. As model size decreases, the model is less robust. While verification results are similar, NNV takes less time to verify than nnenum.

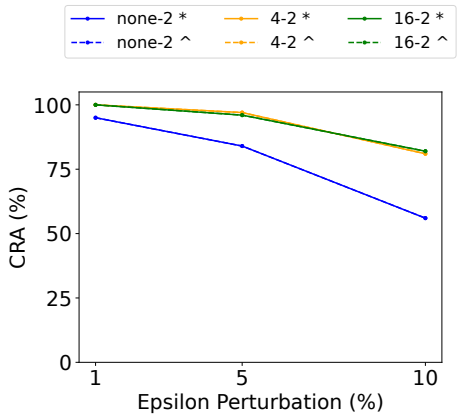
significantly affect CRA for *all* or *discrete & continuous* features, when applied to *discrete* features only, the smaller the model, the less robust it is. This trend is demonstrated by the low blue line (small model) compared to the higher green and orange lines (larger models). NNV and nnenum garner similar CRA results for *discrete* features, but NNV takes less time to verify, as shown by Figure 6b.

(Continuous) The results of using *continuous* features for verification are shown in Figure 7. These results resemble those of *discrete* features, even with the larger perturbation sizes used. This result means that the *discrete* features are more sensitive compared to *continuous* features, as a larger perturbation elicits similar results. When combined (*discrete & continuous*), however, the CRA remains high (Figure 5a). This is likely due to the smaller epsilon values used for *discrete & continuous* features. Regarding general trends in Figure 6a, we see that as the perturbation size increases, larger models are more robust. NNV and nnenum also have similar verification results, but NNV takes less time to verify. The average time to verify for the *none-2* model decreases as the perturbation increases, which makes sense when compared to the decreasing CRA. More samples are being falsified early resulting in a shorter verification time. The average verification time for *4-2* and *16-2*, however, remains approximately the same. This is due to an increase in both falsifications (failing early) and an increase in unknown results (failing late), which is keeping the average time about the same even though the CRA is decreasing.

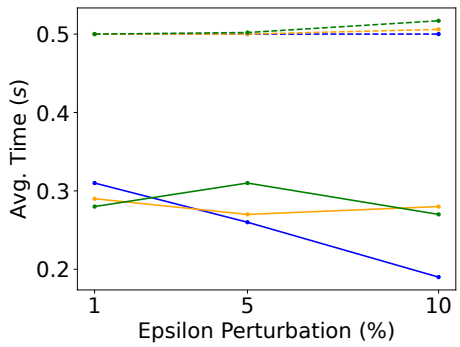
4.2 Malimg

4.2.1 Model Performance. From the training results shown in Table 4, each of the trained family classifiers is effectively able to discern malware families from known malware samples, as indicated by the high accuracy, precision, recall, and F1 scores.

4.2.2 Verification Performance. The results of the Malimg model verification are shown in Figure 8. As the perturbation size increases, the models decrease in CRA, which is especially apparent for the larger models (*4-25* and *16-25*). The larger the model, the more time typically required for each of the verification steps following falsification, as the reachable set for a larger model will be greater than a smaller, linear model. The results in Figure 8 follow this

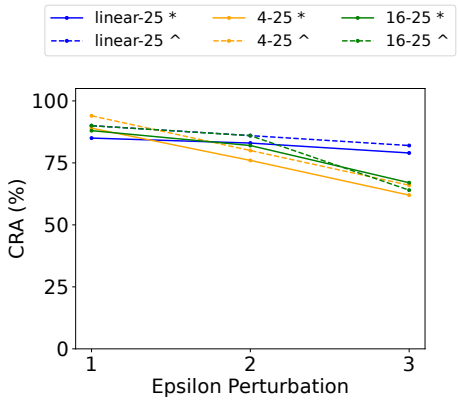


(a) Number Robust

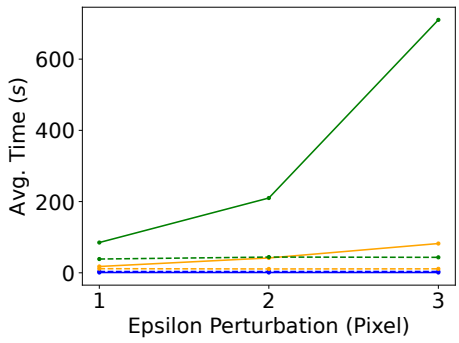


(b) Average Time to Verify

Figure 7: BODMAS (binary) robustness results ($\mathcal{L}_\infty$) for *continuous* features using NNV (*) and nnenum (∧) verification tools. Larger models are more robust as the perturbation size increases. While NNV and nnenum have similar verification results, NNV takes less time to verify.



(a) Number Robust



(b) Average Time to Verify

Figure 8: Malimg (family) robustness results ($\mathcal{L}_\infty$) using NNV (*) and nnenum (∧) verification tools. As perturbation size increases, the CRA decreases for each model. The larger the model, the more time it takes to verify as the perturbation increases.

trend—the larger the model, the more time needed to verify the model, which causes timeouts and a lower CRA value. As a result, smaller models are preferred for this dataset, as they maintain robustness while taking less time to verify.

5 DISCUSSION

As a result of these experiments, we highlight important features for consideration and future study.

Timing: For many of the experiments, NNV resulted in a faster time to verify than nnenum. This is in part due to the different verification methods used by NNV and nnenum. Whereas nnenum is a command line tool requiring only the model and a VNN-LIB file, NNV is a scriptable tool that requires defining the verification steps used during the verification process. Both show the benefit of using each tool: nnenum is a faster process to get started (command line) and NNV is more versatile (adaptable) and, in this case, faster.

Importance of Features: In the BODMAS dataset, making feature specific perturbations makes a significant difference in the robustness evaluations of the model, as shown by the difference between Figure 4a and Figure 5a. More work is therefore necessary to determine meaningful features and meaningful perturbations to those features.

Sample Coverage: The verification sets for both Malimg and BODMAS contain inputs that rarely, if ever, generate a 'robust' outcome. Table 7 shows the per class results of verification using NNV on the samples perturbed with $\epsilon = 2$. *Autorun.K*, *Swizzor.gen!*, and *Swizzor.gen!I* have a low number of robust samples for each trained model. While these classes are underrepresented during training compared to classes like *Allaple.A*, other underrepresented classes do not have similar results. For instance, *Skintrim.N* only has 55 training samples, but images from this class are robustly verified 5/5 times for each model. These results highlight that some classes are more difficult to verify than others. It might be better to evaluate malware classifier models on levels of inputs rather than a conglomerate. For instance, model X is robust to levels 1, 2, and 3 but not to level 4. This might provide more valuable insight into the robustness of a model. This level-specific result will be an interesting problem to explore in future works.

Malimg Class	No. Certified Robust			Train Samples
	linear-25	4-25	16-25	
Adialer.C	5	5	5	97
Agent.FYI	5	5	5	91
Allaple.A	5	5	5	2824
Allaple.L	5	4	4	1491
Alueron.gen!J	5	5	5	173
Autorun.K	0	0	0	81
C2LOP.P	4	2	1	121
C2LOP.gen!g	5	3	4	175
Dialplatform.B	5	5	5	152
Dontovo.A	5	5	5	137
Fakerean	5	5	5	306
Instantaccess	5	5	5	356
Lolyda.AA1	3	5	5	153
Lolyda.AA2	5	5	5	159
Lolyda.AA3	3	5	5	98
Lolyda.AT	5	2	5	134
Malex.gen!J	3	0	3	111
Obfuscator.AD	5	5	5	117
Rbot!gen	5	5	5	133
Skintrim.N	5	5	5	55
Swizzor.gen!E	2	0	0	103
Swizzor.gen!I	0	0	0	107
VB.AT	5	5	5	383
Wintrim.BX	4	4	5	72
Yuner.A	5	5	5	775

Table 7: Per class robustness accuracy on the Malimg verification set for the $\epsilon = 2$ perturbation using NNV. Some classes are more difficult to verify than others, as highlighted by the *Autorun.K, Swizzor.gen!E,* and *Swizzor.gen!I* classes.

6 RELATED WORK

Malware Classification: Previous works have shown the promise of using machine learning models for malware classification. In [15, 26], the authors use data mining to extract meaningful features, and then apply traditional machine learning algorithms. Recent research has shifted to the use of neural networks. In [23], the authors introduce the use of image malware classification, i.e., converting binaries into images and utilizing convolution neural networks (CNNs) for binary and/or family classification. This approach has since been refined by numerous works, including [35].

Semantically Meaningful Perturbations: Semantically meaningful perturbations have been addressed in recent work related to images. In [25], the authors investigate contextually meaningful perturbations to deep neural network image classifiers on 'German Traffic Sign' and 'CIFAR-10' data. In [34], the authors utilize a domain 'deletion' perturbation to medical image pathologies.

Neural Network Verification: The area of DNN verification has increasingly grown in recent years, leading to the development of standard input formats[5] [9] as well as friendly competitions [18, 22], that help compare and evaluate all the recent methods and tools proposed in the community [2, 12, 13, 36, 37]. The majority of these methods focus on verifying FFNN and CNN architectures. These approaches can generally be classified into sound or unsound, and complete or incomplete. Unsound refers to probabilistic analysis such as [30], although they are less common for NN verification than sound approaches. Complete and sound methods refer to algorithms that can precisely analyze whether a given property holds on a model, also referred to as exact methods. However, these type of methods suffer from scalability issues as the exact analysis tend to be very computationally expensive. These are also limited in the type of layers and model architectures they can be applied to. These can be Satisfiability Modulo Theories (SMT) based methods [12, 13], Mixed Integer Linear Program (MILP) based methods [28], reachability analysis methods [32], and others such as branch and bound methods [3]. Incomplete methods refer to sound approaches that present a tradeoff between precision and scalability, which allows a faster computation of a verification problem of larger NNs than sound and complete methods, although it can lead to *unknown* results due to the approximation used. Several of these methods are based on abstract interpretation, some of which have demonstrated to outperform complete methods by orders of magnitude (time wise) [22]. Recent work in [10] has enhanced the abstraction-based verification of neural networks via residual reasoning.

[5]vnnlib: https://www.vnnlib.org

7 CONCLUSION

In this paper, we present a case study to introduce and evaluate the novel formal verification of malware classification for family identification as well as malware identification. Through testing, training, and verification processes, we highlight important current malware verification capabilities as well as areas to be considered moving forward. Through this rigorous analysis, we hope to increase the visibility of this verification domain, bolstering refinement to a safety-critical system that has a drastic impact on our lives every day.

ACKNOWLEDGMENTS

This paper was supported in part by a fellowship award under contract FA9550-21-F-0003 through the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program, sponsored by the Air Force Research Laboratory (AFRL), the Office of Naval Research (ONR), and the Army Research Office (ARO). The material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 2220426 and 2220401, the Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-23-C-0518, and the Air Force Office of Scientific Research (AFOSR) under contract number FA9550-22-1-0019 and FA9550-23-1-0135. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of AFOSR, DARPA, or NSF.

REFERENCES

[1] [n. d.]. 2022 Incident Response Report. https://www.paloaltonetworks.com/unit42/2022-incident-response-report
[2] Stanley Bak. 2021. nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement. In *NASA Formal Methods Symposium*. Springer.

[3] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, M. Pawan Kumar, Jingyue Lu, and Pushmeet Kohli. 2020. Branch and Bound for Piecewise Linear Neural Network Verification. *J. Mach. Learn. Res.* 21, 1, Article 42 (jan 2020), 39 pages.

[4] Nicholas Carlini, Florian Tramer, Krishnamurthy Dj Dvijotham, Leslie Rice, Mingjie Sun, and J Zico Kolter. 2023. (Certified!!) Adversarial Robustness for Free!. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=JLg5aHHv7j

[5] Nicholas Carlini and David Wagner. 2017. Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478* (2017).

[6] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. 2020. On training robust PDF malware classifiers. In *Proceedings of the 29th USENIX Conference on Security Symposium*. 2343–2360.

[7] Christian Cianfarani, Arjun Nitin Bhagoji, Vikash Sehwag, Ben Zhao, Heather Zheng, and Prateek Mittal. 2022. Understanding robust learning through the lens of representation similarities. *Advances in Neural Information Processing Systems* 35 (2022), 34912–34925.

[8] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*. PMLR, 1310–1320.

[9] Stefano Demarchi, Dario Guidotti, Luca Pulina, and Armando Tacchella. 2023. Supporting Standardization of Neural Networks Verification with VNNLIB and CoCoNet. In *Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems (Kalpa Publications in Computing, Vol. 16)*, Nina Narodytska, Guy Amir, Guy Katz, and Omri Isac (Eds.). EasyChair, 47–58. https://doi.org/10.29007/5pdh

[10] Yizhak Yisrael Elboher, Elazar Cohen, and Guy Katz. 2022. Neural network verification using residual reasoning. In *Software Engineering and Formal Methods: 20th International Conference, SEFM 2022, Berlin, Germany, September 26–30, 2022, Proceedings*. Springer, 173–189.

[11] Zhuoqun Huang, Neil G Marchant, Keane Lucas, Lujo Bauer, Olga Ohrimenko, and Benjamin I. P. Rubinstein. 2023. RS-Del: Edit Distance Robustness Certificates for Sequence Classifiers via Randomized Deletion. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=ffFcRPpnWx

[12] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[13] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 443–452.

[14] Hae-Jung Kim. 2018. Image-based malware classification using convolutional neural network. In *Advances in Computer Science and Ubiquitous Computing: CSA-CUTE 17*. Springer, 1352–1357.

[15] Jeremy Z Kolter and Marcus A Maloof. 2004. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 470–478.

[16] David Kornish, Justin Geary, Victor Sansing, Soundararajan Ezekiel, Larry Pearlstein, and Laurent Njilla. 2018. Malware classification using deep convolutional neural networks. In *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, 1–6.

[17] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 656–672.

[18] Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Xin Chen, Jiameng Fan, Marcelo Forets, Chao Huang, Taylor T Johnson, Tobias Ladner, Wenchao Li, Christian Schilling, and Qi Zhu. 2022. ARCH-COMP22 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. In *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22) (EPiC Series in Computing, Vol. 90)*, Goran Frehse, Matthias Althoff, Erwin Schoitsch, and Jeremie Guiochet (Eds.). EasyChair, 142–184.

[19] Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. 2023. NNV 2.0: The Neural Network Verification Tool. In *35th International Conference on Computer-Aided Verification (CAV)*.

[20] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJzIBfZAb

[21] Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, and Taylor Johnson. 2022. Reachability Analysis of a General Class of Neural Ordinary Differential Equation. In *Proceedings of the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2022), Co-Located with CONCUR, FMICS, and QEST as part of CONFEST 2022*. Warsaw, Poland.

[22] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. 2022. The Third International Verification of Neural Networks Competition (VNN-COMP 2022): Summary and Results.

[23] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. 2011. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*. 1–7.

[24] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 582–597.

[25] Colin Paterson, Haoze Wu, John Grese, Radu Calinescu, Corina S Păsăreanu, and Clark Barrett. 2021. Deepcert: Verification of contextually relevant robustness for neural network image classifiers. In *Computer Safety, Reliability, and Security: 40th International Conference, SAFECOMP 2021, York, UK, September 8–10, 2021, Proceedings 40*. Springer, 3–17.

[26] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. 2000. Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 38–49.

[27] Ajay Singh, Anand Handa, Nitesh Kumar, and Sandeep Kumar Shukla. 2019. Malware classification using image representation. In *Cyber Security Cryptography and Machine Learning: Third International Symposium, CSCML 2019, Beer-Sheva, Israel, June 27–28, 2019, Proceedings 3*. Springer, 75–92.

[28] Vincent Tjeng, Kai Xiao, and Russ Tedrake. 2017. Evaluating Robustness of Neural Networks with Mixed Integer Programming. *arXiv preprint arXiv:1711.07356* (2017).

[29] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020. Verification of Deep Convolutional Neural Networks Using ImageStars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer.

[30] Hoang-Dung Tran, Sungwoo Choi, Hideki Okamoto, Bardh Hoxha, Georgios Fainekos, and Danil Prokhorov. 2023. Quantitative Verification for Neural Networks Using ProbStars. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control* (San Antonio, TX, USA) *(HSCC '23)*. Association for Computing Machinery, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3575870.3587112

[31] Hoang Dung Tran, SungWoo Choi, Tomoya Yamaguchi, Bardh Hoxha, and Danil Prokhorov. 2023. Verification of Recurrent Neural Networks using Star Reachability. In *The 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*.

[32] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Xiaodong Yang, Nathaniel P. Hamilton, Diego Manzanas Lopez, Stanley Bak, and Taylor T. Johnson. 2021. Robustness Verification of Semantic Segmentation Neural Networks using Relaxed Reachability. In *33rd International Conference on Computer-Aided Verification (CAV)*. Springer.

[33] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In *32nd International Conference on Computer-Aided Verification (CAV)*.

[34] Hristina Uzunova, Jan Ehrhardt, Timo Kepp, and Heinz Handels. 2019. Interpretable explanations of black box classifiers applied on medical images by meaningful perturbations using variational autoencoders. In *Medical Imaging 2019: Image Processing*, Vol. 10949. SPIE, 264–271.

[35] Danish Vasan, Mamoun Alazab, Sobia Wassan, Babak Safaei, and Qin Zheng. 2020. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Computers & Security* 92 (2020), 101748.

[36] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1599–1614.

[37] 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 neural network robustness verification. *Advances in Neural Information Processing Systems* 34 (2021), 29909–29921.

[38] Greg Yang, Tony Duan, J Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. 2020. Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning*. PMLR, 10693–10705.

[39] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2021. BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware. In *4th Deep Learning and Security Workshop*.