Verifying Properties of Binarized Deep Neural Networks

Nina Narodytska
VMware Research
Palo Alto, USA
n.narodytska@vmware.com

Shiva Kasiviswanathan*
Amazon ML
Palo Alto, USA
kasivisw@gmail.com

Leonid Ryzhyk
VMware Research
Palo Alto, USA
lryzhyk@vmware.com

Mooly Sagiv
VMware Research
Palo Alto, USA
msagiv@vmware.com

Toby Walsh**
Data61, UNSW
Sydney

Abstract

Understanding properties of deep neural networks is an important challenge in deep learning. In this paper, we take a step in this direction by proposing a rigorous way of verifying properties of a popular class of neural networks, Binarized Neural Networks, using the well-developed means of Boolean satisfiability. Our main contribution is a construction that creates a representation of a binarized neural network as a Boolean formula. Our encoding is the first *exact* Boolean representation of a deep neural network. Using this encoding, we leverage the power of modern SAT solvers along with a proposed counterexample-guided search procedure to verify various properties of these networks. A particular focus will be on the critical property of robustness to adversarial perturbations. For this property, our experimental results demonstrate that our approach scales to medium-size deep neural networks used in image classification tasks. To the best of our knowledge, this is the first work on verifying properties of deep neural networks using an exact Boolean encoding of the network.

1 Introduction

Deep neural networks have become ubiquitous in machine learning with applications ranging from computer vision to speech recognition and natural language processing. Neural networks demonstrate excellent performance in many practical problems, often beating specialized algorithms for these problems, which led to their rapid adoption in industrial applications. With such a wide adoption, important questions arise regarding our understanding of these neural networks: How robust are these networks to perturbations of inputs? How critical is the choice of one architecture over the other? Does certain ordering of transformations matter? Recently, a new line of research on understanding neural networks has emerged that look into a wide range of such questions, from interpretability of neural networks to verifying their properties (Bau et al. 2017; Szegedy et al. 2014; Bastani et al. 2016; Huang et al. 2017; Katz et al. 2017).

In this work we focus on an important class of neural networks: Binarized Neural Networks (BNNs) (Hubara et al. 2016). These networks have a number of important features that are useful in resource constrained environments, like embedded devices or mobile phones. Firstly, these networks are memory efficient, as their parameters are primarily binary. Secondly, they are computationally efficient as all activations are binary, which enables the use of specialized algorithms for fast binary matrix multiplication. These networks have been shown to achieve performance comparable to traditional deep networks that use floating point precision on several standard datasets (Hubara et al. 2016). Recently, BNNs have been deployed for various embedded applications ranging from image classification (McDanel, Teerapittayanon, and Kung 2017) to object detection (Kung et al. 2017).

The goal of this work is to analyze properties of binarized networks through the lens of Boolean satisfiability (SAT). Our main contribution is a procedure for constructing a SAT encoding of a BNN. An important feature of our encoding is that it is *exact* and does not rely on any approximations to the network structure. This means that this encoding allows us to investigate properties of BNNs by studying similar properties in the SAT domain, and the mapping of these properties back from the SAT to the neural network domain is exact. To the best of our knowledge, this is the first work on verifying properties of deep neural networks using an exact Boolean encoding of a network. In our construction, we exploit attributes of BNN's, both *functional*, e.g., most parameters of these networks are binary, and *structural*, e.g., the modular structure of these networks. While these encodings could be directly handled by modern SAT solvers, we show that one can exploit the structure of these encodings to solve the resulting SAT formulas more efficiently based on the idea of *counterexample-guided* search (Clarke et al. 2000).

While our encoding could be used to study many properties of BNNs, in this paper we focus on important properties of network *robustness* and *equivalence*.

(a) Deep neural networks have been shown susceptible to crafted adversarial perturbations which force misclassification of the inputs. Adversarial inputs can be used to subvert fraud detection, malware detection, or mislead autonomous navigation systems (Papernot et al. 2016; Grosse et al. 2016) and pose a serious security risk (e.g., consider an adversary that can fool an autonomous driving system into not following posted traffic signs). Therefore, *certifiably* verifying robustness of these networks to adversarial perturbation is a question of paramount practical importance. Using a SAT encoding, we can certifiably establish whether or not a BNN

*Part of the work done while the author was at Samsung Research America.

**Funded by the European Research Council under the EU Horizon 2020 programme via grant AMPLIFY 670077.

is robust to adversarial perturbation on a particular image.

(b) Problems of verifying whether two networks are equivalent in their operations regularly arise when dealing with network alterations (such as that produced by *model reduction* operations (Reagen et al. 2017)) and input preprocessing. Again, with our SAT encodings, we can check for network equivalence or produce instances where the two networks differ in their operation.

Experimentally, we show that our techniques can verify properties of medium-sized BNNs. In Section 8, we present a set of experiments on the MNIST dataset and its variants. For example, for a BNN with five fully connected layers, we are able to prove the absence of an adversarial perturbation or find such a perturbation for up to 95% of considered images.

2 Preliminaries

Notation. We denote $[m] = \{1, \ldots, m\}$. Vectors are in column-wise fashion, denoted by boldface letters. For a vector $\mathbf{v} \in \mathbb{R}^m$, we use $(v_1, \ldots, v_m)$ to denote its m elements. For $p \geq 1$, we use $\|\mathbf{v}\|_p$ to denote the L_p-norm of $\mathbf{v}$. For a Boolean CNF (Conjunctive Normal Form) formula A, let $\text{vars}(A)$ denote the set of variables in A. We say A is *unsatisfiable* if there exists no assignment to the variables in $\text{vars}(A)$ that evaluates the formula to true, otherwise we say A is *satisfiable*. Let A and B be Boolean formulas. We denote $\bar{A}$ the negation of A. We say that A *implies* B ($A \Rightarrow B$) iff $\bar{A} \vee B$ is satisfiable and A is *equivalent* to B ($A \Leftrightarrow B$) iff $A \Rightarrow B$ and $B \Rightarrow A$.

Next, we define the supervised image classification problem that we focus on. We are given a set of training images drawn from an unknown distribution ν over $\mathbb{Z}^n$, where n here represents the "size" of individual images. Each image is associated with a label generated by an unknown function $\mathrm{L} : \mathbb{Z}^n \to [s]$, where $[s] = \{1, \ldots, s\}$ is a set of possible labels. During the training phase, given a labeled training set, the goal is to learn a neural network classifier that can be used for inference: given a new image drawn from the same distribution ν, the classifier should predict its true label. During the inference phase, the network is *fixed*. In this work, we study properties of such fixed networks generated post training.

Properties of Neural Networks

In this section, we define several important properties of neural networks, ranging from robustness to properties related to network structure. As the properties defined in this section are not specific to BNNs, we consider a general feedforward neural network denoted by F. Let $\mathrm{F}(\mathbf{x})$ represent the output of F on input $\mathbf{x}$ and $\ell_{\mathbf{x}} = \mathrm{L}(\mathbf{x})$ be the ground truth label of $\mathbf{x}$.

Adversarial Network Robustness. Robustness is an important property that guards the network against adversarial tampering of its outcome by perturbing the inputs. There are now many techniques for generating adversarial inputs, e.g., see (Szegedy et al. 2014; Goodfellow, Shlens, and Szegedy 2015; Moosavi-Dezfooli, Fawzi, and Frossard 2016). Therefore, a natural question is to understand how susceptible a network is to any form of adversarial perturbation (Goodfellow, Shlens, and Szegedy 2015; Moosavi-Dezfooli, Fawzi, and Frossard 2016; Bastani et al. 2016). Informally, a network is robust on an input if small perturbations to the input does not lead to misclassification. More formally,

Definition 1 (Adversarial Robustness). [1] *A feedforward neural network* F *is* (ϵ, p)*-robust for an input* $\mathbf{x}$ *if there does not exist* τ, $\|\tau\|_p \leq \epsilon$, *such that* $\mathrm{F}(\mathbf{x} + \tau) \neq \ell_{\mathbf{x}}$.

Since for any fixed vector, the L_p-norms monotonically decrease as we increase p, if a network is (ϵ, p)-robust then it is also (ϵ, p')-robust for all $p' > p$. The case of $p = \infty$, which bounds the maximum perturbation applied to each entry in $\mathbf{x}$, is especially interesting and has been considered frequently in the literature on adversarial attacks in deep learning (Goodfellow, Shlens, and Szegedy 2015; Bastani et al. 2016; Katz et al. 2017).

Another popular definition of robustness comes from the notion of *universal adversarial* perturbation as defined by (Moosavi-Dezfooli et al. 2016). Intuitively, a universal adversarial perturbation is one that leads to misclassification on most (say, ρ-fraction) of the inputs in a set of images. Absence of such a perturbation is captured in the following definition of robustness. Let S denote a set of images.

Definition 2 (Universal Adversarial Robustness). *A feedforward neural network* F *is* (ϵ, p, ρ)*-universally robust for a set of inputs in* S *if there does not exist* τ, $\|\tau\|_p \leq \epsilon$, *such that* $\sum_{\mathbf{x}_i \in S} \mathbb{1}_{\mathrm{F}(\mathbf{x}_i + \tau) \neq l_{\mathbf{x}_i}} \geq \rho |S|$.

Network Equivalence. Similar to robustness, a property that is commonly verified is that of *equivalence* of networks. Informally, two networks F_1 and F_2 are equivalent if these networks generate same outputs on all inputs drawn from the domain. Let $\mathcal{X}$ denote the domain from which inputs are drawn. In the case of images, $\mathcal{X} = \mathbb{Z}^n$.

Definition 3 (Network Equivalence). *Two feedforward neural networks* F_1 *and* F_2 *are equivalent if for all* $\mathbf{x} \in \mathcal{X}$, $\mathrm{F}_1(\mathbf{x}) = \mathrm{F}_2(\mathbf{x})$.

We describe two common scenarios where such equivalence problem arises in practice.

(1) Network Alteration: Consider a scenario where a part of the trained network has been altered to form a new network. This change could arise due to model reduction operations that are commonly performed on deep networks to make them amenable for execution on resource-constrained devices (Reagen et al. 2017) or they could arise from other sources of noise including adversarial corruption of the network. The question now is whether the altered network is equivalent to the original network?

(2) Augmentation Reordering: Consider a scenario where an input is preprocessed (augmented) before it is supplied to a network. Examples of such preprocessing include geometrical transformations, blurring, etc. Let $f : \mathcal{X} \to \mathcal{X}$ and $g : \mathcal{X} \to \mathcal{X}$ be two transformation functions (this extends to more transformation functions too). We want to know how sensitive the network is to the order of applications of f and g. For example, given a network F, let F_1 be the network

[1] The definition naturally extends to a collection of inputs.

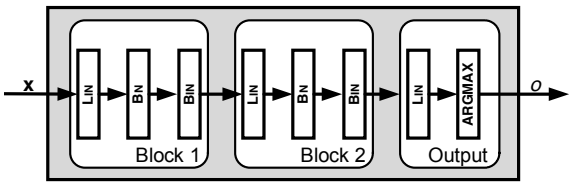

Figure 1: A schematic view of a binarized neural network. The internal blocks also have an additional hard tanh layer.

that applies $f \circ g$ on the input followed by F, and F_2 be the network that applies $g \circ f$ on the input followed by F. The question now is whether F_1 is equivalent to F_2?

3 Binarized Neural Networks

A binarized neural network (BNN) is a feedforward network where weights and activations are predominantly binary (Hubara et al. 2016). It is convenient to describe the structure of a BNN in terms of composition of blocks of layers rather than individual layers. Each block consists of a collection of linear and non-linear transformations. Blocks are assembled sequentially to form a BNN.

Internal Block. Each internal block (denoted as BLK) in a BNN performs a collection of transformations over a binary input vector and outputs a binary vector. While the input and output of a BLK are binary vectors, the internal layers of BLK can produce real-valued intermediate outputs. A common construction of an internal BLK (taken from (Hubara et al. 2016)) is composed of three main operations:[2] a linear transformation (LIN), batch normalization (BN), and binarization (BIN). Table 1 presents the formal definition of these transformations. The first step is a linear (affine) transformation of the input vector. The linear transformation can be based on a fully connected layer or a convolutional layer. The linear transformation is followed by a scaling which is performed with a batch normalization operation (Ioffe and Szegedy 2015). Finally, a binarization is performed using the sign function to obtain a binary output vector.[3] Figure 1 shows two BLKs connected sequentially.

Output Block. The output block (denoted as O) produces the classification decision for a given image. It consists of two layers (see Table 1). The first layer applies a linear (affine) transformation that maps its input to a vector of integers, one for each output label class. This is followed by a ARGMAX layer, which outputs the index of the largest entry in this vector as the predicted label.

Network of Blocks. BNN is a deep feedforward network formed by assembling a sequence of internal blocks and an output block. Suppose we have $d-1$ blocks, $\text{BLK}_1, \ldots, \text{BLK}_{d-1}$ that are placed consecutively, so the output of a block is an input to the next block in the list. Let $\mathbf{x}_k$ be the input to BLK_k and $\mathbf{x}_{k+1}$ be its output. The input of the first block is the input of the network. We assume that the input of the network is a vector of integers, which is true for the image classification task if images are in the standard RGB format. Note that these integers can be encoded with binary values $\{-1, 1\}$ using a standard encoding. Therefore, we keep notations uniform for all layers by assuming that inputs are all binary. The output of the last layer, $\mathbf{x}_d \in \{-1, 1\}^{n_d}$, is passed to the output block O to obtain the label.

Definition 4. *A binarized neural network* BNN $: \{-1, 1\}^n \to [s]$ *is a feedforward network that is composed of* d *blocks,* $\text{BLK}_1, \ldots, \text{BLK}_{d-1}$, O. *Formally, given an input* $\mathbf{x}$, $\text{BNN}(\mathbf{x}) = \text{O}(\text{BLK}_{d-1}(\ldots \text{BLK}_1(\mathbf{x}) \ldots))$.

4 Encodings of Binarized Networks

In this section, we consider encodings of BNNs into Boolean formulae. To this end, we encode the two building blocks (BLK and O) of the network into SAT. The encoding of the BNN is a conjunction of encodings of its blocks. Let $\text{BINBLK}_k(\mathbf{x}_k, \mathbf{x}_{k+1})$ be a Boolean function that encodes the kth block (BLK_k) with an input $\mathbf{x}_k$ and an output $\mathbf{x}_{k+1}$. Similarly, let $\text{BINO}(\mathbf{x}_d, o)$ be a Boolean function that encodes O that takes an input $\mathbf{x}_d$ and outputs o. The entire BNN on input $\mathbf{x}$ can be encoded as a Boolean formula (with $\mathbf{x}_1 = \mathbf{x}$):

$$\left(\bigwedge_{k=1}^{d-1} \text{BINBLK}_k(\mathbf{x}_k, \mathbf{x}_{k+1})\right) \wedge \text{BINO}(\mathbf{x}_d, o).$$

We consider several encodings of BINBLK and BINO starting with a simple MILP encoding, which is refined to get an ILP encoding, which is further refined to the final SAT encoding.

Mixed Integer Linear Program Encoding

We start with a Mixed Integer Linear Programming (MILP) encoding of BLK and O blocks. Our MILP encoding has the same flavor as other encodings in the literature for non-binarized networks, e.g., (Bastani et al. 2016; Pulina and Tacchella 2012) (see Section 7).

Encoding of BLK_k**.** We encode each layer in BLK_k to MILP separately. Let $\mathbf{a}_i$ be the ith row of the matrix A_k. Let $\mathbf{x}_k \in \{-1, 1\}^{n_k}$ denote the input to BLK_k.

Linear Transformation. The first layer we consider is the linear layer (LIN). In the following, for simplicity, we focus on the linear transformation applied by a fully connected layer but note that a similar encoding can also be done for a convolution layer (as convolutions are also linear operations). The linear constraint in this case is simply,

$$y_i = \langle \mathbf{a}_i, \mathbf{x}_k \rangle + b_i, i = 1, \ldots, n_{k+1}, \quad (1)$$

where $\mathbf{y} = (y_1, \ldots, y_{n_{k+1}})$ is a vector in $\mathbb{R}^{n_{k+1}}$ because the bias part, $\mathbf{b}$, is a vector of real values.

Batch Normalization. The second layer is the batch normalization layer (BN) that takes the output of the linear layer as

[2] In the training phase, there is an additional *hard tanh* layer after batch normalization layer that is omitted in the inference phase (Hubara et al. 2016).

[3] A BLK may also contain a max pooling operation to perform dimensionality reduction, which can be simulated using a convolutional layer with an appropriately chosen stride (Springenberg et al. 2015).

Structure of kth Internal block, $\text{BLK}_k : \{-1,1\}^{n_k} \to \{-1,1\}^{n_{k+1}}$ on input $\mathbf{x}_k \in \{-1,1\}^{n_k}$	
LIN	$\mathbf{y} = A_k\mathbf{x}_k + \mathbf{b}_k$, where $A_k \in \{-1,1\}^{n_{k+1}\times n_k}$ and $\mathbf{b}_k \in \mathbb{R}^{n_{k+1}}$
BN	$z_i = \alpha_{k_i}\left(\frac{y_i-\mu_{k_i}}{\sigma_{k_i}}\right) + \gamma_{k_i}$, where $\mathbf{y} = (y_1,\ldots,y_{n_{k+1}})$, and $\alpha_{k_i}, \gamma_{k_i}, \mu_{k_i}, \sigma_{k_i} \in \mathbb{R}$. Assume $\sigma_{k_i} > 0$.
BIN	$\mathbf{x}_{k+1} = \text{sign}(\mathbf{z})$ where $\mathbf{z} = (z_1,\ldots,z_{n_{k+1}}) \in \mathbb{R}^{n_{k+1}}$ and $\mathbf{x}_{k+1} \in \{-1,1\}^{n_{k+1}}$
Structure of Output Block, $\text{O} : \{-1,1\}^{n_d} \to [s]$ on input $\mathbf{x}_d \in \{-1,1\}^{n_d}$	
LIN	$\mathbf{w} = A_d\mathbf{x}_d + \mathbf{b}_d$, where $A_d \in \{-1,1\}^{s\times n_d}$ and $\mathbf{b}_d \in \mathbb{R}^s$
ARGMAX	$o = \text{argmax}(\mathbf{w})$, where $o \in [s]$

Table 1: Structure of internal and outputs blocks which stacked together form a binarized neural network. In the training phase, there might be an additional *hard tanh* layer after batch normalization. A_k and b_k are parameters of the LIN layer, whereas $\alpha_{k_i}, \gamma_{k_i}, \mu_{k_i}, \sigma_{k_i}$ are parameters of the BN layer. The μ's and σ's correspond to mean and standard deviation computed in the training phase. The BIN layer is parameter free.

an input. By definition, we have

$$z_i = \alpha_{k_i}\left(\frac{y_i-\mu_{k_i}}{\sigma_{k_i}}\right) + \gamma_{k_i}, \quad i = 1,\ldots,n_{k+1}, \text{ or}$$
$$\sigma_{k_i} z_i = \alpha_{k_i} y_i - \alpha_{k_i}\mu_{k_i} + \sigma_{k_i}\gamma_{k_i}, \quad i = 1,\ldots,n_{k+1}. \quad (2)$$

The above equation is a linear constraint.

Binarization. For the BIN operation, which implements a sign function, we need to deal with conditional constraints.

$$z_i \geq 0 \Rightarrow v_i = 1, \quad i = 1,\ldots,n_{k+1}, \quad (3)$$
$$z_i < 0 \Rightarrow v_i = -1, \quad i = 1,\ldots,n_{k+1}. \quad (4)$$

Since these constraints are implication constraints, we can use a standard trick with the big-M formulation to encode them (Griva, Nash, and Sofer 2009). Define, $\mathbf{x}_{k+1} = (v_1,\ldots,v_{n_{k+1}})$.

Example 1. *Consider an internal block with two inputs and one output. Suppose we have the following parameters:* $A_k = [1,-1]$, $b_k = [-0.5]$, $\alpha_k = [0.12]$, $\mu_k = [-0.1]$, $\sigma_k = [2]$ *and* $\gamma_k = [0.1]$. *First, we apply the linear transformation:* $y_1 = x_{k_1} - x_{k_2} - 0.5$. *Second, we apply batch normalization:* $2z_1 = 0.12y_1 + 0.12 \times 0.1 + 2 \times 0.1$. *Finally, we apply binarization. We get* $z_1 \geq 0 \Rightarrow v_1 = 1$, *and* $z_1 < 0 \Rightarrow v_1 = -1$. *In this case,* $\mathbf{x}_{k+1}$ *is just* v_1.

Encoding of O. For the O block, encoding of the linear layer is straightforward as in the BLK case:

$$w_i = \langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i, \quad i = 1,\ldots,s,$$

where $\mathbf{a}_i$ now represents the ith column in A_d and $\mathbf{w} = (w_1,\ldots,w_s)$. To encode ARGMAX, we need to encode an ordering relation between w_i's. We assume that there is a total order over w_i's for simplicity. We introduce a set of Boolean variables d_{ij}'s such that

$$w_i \geq w_j \Leftrightarrow d_{ij} = 1, \quad i,j = 1,\ldots,s.$$

Finally, we introduce an output variable o and use more implication constraints.

$$\sum_{j=1}^{s} d_{ij} = s \Rightarrow o = j, \quad i = 1,\ldots,s. \quad (5)$$

Example 2. *Consider an output block with two inputs and two outputs. Suppose we have the following parameters for this block* $A_d = [1,-1;-1,1]$ *and* $b = [-0.5, 0.2]$. *First, we perform encoding of the linear transformation. We get constraints* $w_1 = x_{d_1} - x_{d_2} - 0.5$ *and* $w_2 = -x_{d_1} + x_{d_2} + 0.2$. *As we have only two outputs, we introduce four Boolean variables* d_{ij}, $i,j = 1,2$. *Note that* d_{11} *and* d_{22} *are always true. So, we only need to consider non-diagonal variables. Hence, we have the following constraints:* $w_1 \geq w_2 \Leftrightarrow d_{12} = 1$ *and* $w_2 < w_1 \Leftrightarrow d_{21} = 1$. *Finally, we compute the output* o *of the neural network as:* $d_{11} + d_{12} = 2 \Rightarrow o = 1$ *and* $d_{21} + d_{22} = 2 \Rightarrow o = 2$.

ILP Encoding

In this section, we show how we can get rid of real-valued variables in the MILP encoding to get a pure ILP encoding which is smaller compared to the MILP encoding.

Encoding of BLK_k. As the input and the output of BLK_k are integer (binary) values, both $\mathbf{z}$ and $\mathbf{y}$ are functional variables of $\mathbf{x}_k$. Hence, we can substitute them in (3) and in (4) based on (1) and (2) respectively. We get that

$$\frac{\alpha_{k_i}}{\sigma_{k_i}}(\langle \mathbf{a}_i, \mathbf{x}_k \rangle + b_i) - \frac{\alpha_{k_i}}{\sigma_{k_i}}\mu_{k_i} + \gamma_{k_i} \geq 0 \Rightarrow v_i = 1.$$

Assume $\alpha_{k_i} > 0$, then this translates to

$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq -\frac{\sigma_{k_i}}{\alpha_{k_i}}\gamma_{k_i} + \mu_{k_i} - b_i \Rightarrow x'_i = 1.$$

Consider, the left part of above equation. Now, we notice that the left-hand of the equation ($\langle \mathbf{a}_i, \mathbf{x}_k \rangle$) has an integer value, as $\mathbf{a}_i$ and $\mathbf{x}_k$ are binary vectors. Hence, the right-hand side, which is a real value can be rounded safely. Define $C_i = \lceil -\frac{\sigma_{k_i}}{\alpha_{k_i}}\gamma_{k_i} + \mu_{k_i} - b_i \rceil$. Then we can use the following implication constraint to encode (3) and (4).

$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq C_i \Rightarrow v_i = 1, \quad i = 1,\ldots,n_{k+1}, \quad (6)$$
$$\langle \mathbf{a}_i, \mathbf{x}_k \rangle < C_i \Rightarrow v_i = -1, \quad i = 1,\ldots,n_{k+1}. \quad (7)$$

If $\alpha_{k_i} < 0$, then the idea is again similar, but now we define $C_i = \lfloor -\frac{\sigma_{k_i}}{\alpha_{k_i}}\gamma_{k_i} + \mu_{k_i} - b_i \rfloor$, and use this value of C_i in (6) and (7). If $\alpha_{k_i} = 0$, then we can eliminate these constraints and set v_i's based on the sign of γ_{k_i}'s.

Example 3. *Consider the internal block from Example 1. Following our transformation, we get the following constraints:*

$$x_{k_1} - x_{k_2} \geq \lceil -2 \times 0.1/0.12 - 0.1 + 0.5 \rceil \Rightarrow v_1 = 1,$$
$$x_{k_1} - x_{k_2} < \lceil -2 \times 0.1/0.12 - 0.1 + 0.5 \rceil \Rightarrow v_1 = -1.$$

Encoding of O**.** Next, we consider the output block. As with the MILP encoding, we introduce the Boolean variables d_{ij} but avoid using the intermediate variables w_i's. This translates to:

$$\langle \mathbf{a}_i, \mathbf{x}_d \rangle + b_i \geq \langle \mathbf{a}_j, \mathbf{x}_d \rangle + b_j \Leftrightarrow d_{ij} = 1, \quad i, j = 1, \ldots, s.$$

where $\mathbf{a}_i$ and $\mathbf{a}_j$ denote the ith and jth rows in the matrix A_d. The above constraints can be reexpressed as:

$$\langle \mathbf{a}_i - \mathbf{a}_j, \mathbf{x}_d \rangle \geq \lceil b_j - b_i \rceil \Leftrightarrow d_{ij} = 1, i, j = 1, \ldots s. \quad (8)$$

The rounding is sound as A_d and $\mathbf{x}_d$ have only integer (binary) entries. Constraints from (5) can be reused as all variables there are integers.

Example 4. *Consider the output block from Example 2 and constraints for d_{12} (encoding for d_{21} is similar). We follow the transformation above to get: $x_{d_1} - x_{d_2} - 0.5 \geq -x_{d_1} + x_{d_2} + 0.2 \Leftrightarrow d_{12} = 1$, which can be reexpressed as: $x_{d_1} - x_{d_2} \geq \lceil 0.7/2 \rceil \Leftrightarrow d_{12} = 1$.*

SAT (CNF) Encoding

Our next step will be to go from the ILP to a pure SAT encoding. A trivial way to encode the BLK and O blocks into CNF is to directly translate their ILP encoding defined above into SAT, but this is inefficient as the resulting encoding will be very large and the SAT formula will blow up even with logarithmic encoding of integers. In this section, we will further exploit properties of binarized neural networks to construct a more compact SAT encoding. We first recall the definition of *sequential counters* from (Sinz 2005) that are used to encode cardinality constraints.

Sequential Counters. Consider a cardinality constraint: $\sum_{i=1}^{m} l_i \geq C$, where $l_i \in \{0, 1\}$ is a Boolean variable and C is a constant. Then the sequential counter encoding, that we denote as $\text{SQ}(\mathbf{l}, C)$ with $\mathbf{l} = (l_1, \ldots, l_m)$, is the following CNF formula:

$$\begin{aligned}
\bar{l}_1 \vee r_{(1,1)} \quad & \wedge \\
\bar{r}_{1,j}, \quad j \in \{2, \ldots, C\} \quad & \wedge \\
\bar{l}_i \vee r_{(i,1)} \quad & \wedge \\
\bar{r}_{i-1,1} \vee r_{(i,1)} \quad & \wedge \\
\bar{l}_i \vee \bar{r}_{(i-1,j-1)} \vee r_{(i,j)}, \quad j \in \{2, \ldots, C\} \quad & \wedge \\
\bar{r}_{i-1,j} \vee r_{(i,j)}, \quad j \in \{2, \ldots, C\} \quad & \wedge \\
\bar{l}_i \vee \bar{r}_{(i-1,C)} \quad & \wedge \\
\bar{l}_m \vee \bar{r}_{(m-1,C)}, \quad & \quad (9)
\end{aligned}$$

where $i \in \{2, \ldots, m\}$. Essentially, this encoding computes the cumulative sums $\sum_{i=1}^{p} l_i$ and the computation is performed in unary, i.e., the auxiliary Boolean variable $r_{(j,p)}$ is true iff $\sum_{i=1}^{j} l_i \geq p$. In particular, the auxiliary variable, $r_{(m,C)}$ encodes whether $\sum_{i=1}^{m} l_i$ is greater or equal to C.

Encoding of BLK_k**.** Looking at the constraint in (6) (constraint in (7) can be handed similarly), we note that the only type of constraint that we need to encode into SAT is a constraint of the form $\langle \mathbf{a}_i, \mathbf{x}_k \rangle \geq C_i \Rightarrow v_i = 1$. Moreover, we recall that all values in $\mathbf{a}_i$'s are binary and $\mathbf{x}_k$ is a binary vector. Let us consider the left part of (6) and rewrite it using the summation operation: $\sum_{j=1}^{n_k} a_{ij} x_{k_j} \geq C_i$, where a_{ij} is the (i, j)th entry in A_k and $\mathbf{x}_{k_j}$ is the j entry in $\mathbf{x}_k$. Next, we perform a variable replacement where we replace $x_{k_j} \in \{-1, 1\}$ with a Boolean variable $x_{k_j}^{(b)} \in \{0, 1\}$, with $x_{k_j} = 2x_{k_j}^{(b)} - 1$. Then, we can rewrite the summation as,

$$\sum_{j=1}^{n_k} a_{ij}(2x_{k_j}^{(b)} - 1) \geq C_i. \quad (10)$$

Denote $\mathbf{a}_i^+ = \{j \mid a_{ij} = 1\}$ and $\mathbf{a}_i^- = \{j \mid a_{ij} = -1\}$. Now (10) can be rewritten as:

$$\begin{aligned}
\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} x_{k_j}^{(b)} &\geq \lceil C_i/2 + \sum_{j=1}^{n_k} a_{ij}/2 \rceil, \\
\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} - \sum_{j \in \mathbf{a}_i^-} (1 - \bar{x}_{k_j}^{(b)}) &\geq C_i', \\
\sum_{j \in \mathbf{a}_i^+} x_{k_j}^{(b)} + \sum_{j \in \mathbf{a}_i^-} \bar{x}_{k_j}^{(b)} &\geq C_i' + |\mathbf{a}_i^-|, \quad (11)
\end{aligned}$$

where $C_i' = \lceil C_i/2 + \sum_{j=1}^{n_k} a_{ij}/2 \rceil$. Let $l_{k_j} = x_{k_j}^{(b)}$ if $j \in \mathbf{a}_i^+$ and let $l_{k_j} = \bar{x}_{k_j}^{(b)}$ if $j \in \mathbf{a}_i^-$. Let $D_i = C_i' + |\mathbf{a}_i^-|$. Then (11) can be reexpressed as: $\sum_{j=1}^{n_k} l_{k_j} \geq D_i$, which is a cardinality constraint. We can rewrite the constraints in (6) and (7) as:

$$\sum_{j=1}^{n_k} l_{k_j} \geq D_i \Leftrightarrow v_i^{(b)} = 1, \quad i = 1, \ldots, n_{k+1}, \quad (12)$$

where $v_i^{(b)} \in \{0, 1\}$ is such that $v_i^{(b)} \Rightarrow v_i = 1$ and $\bar{v}_i^{(b)} \Rightarrow v_i = -1$. Notice that (12) is a cardinality constraint conjoined with an equivalence constraint. Let $\text{SQ}(\mathbf{l}_k, D_i)$ denote the sequential counter encoding of (12) with auxiliary variables $r_{i_{(1,1)}}, \ldots, r_{i_{(n_k, D_i)}}$. Then, $r_{i_{(n_k, D_i)}} \Leftrightarrow v_i^{(b)}$.

Putting everything together, we define the CNF formula $\text{BINBLK}_k(\mathbf{x}_k, \mathbf{x}_{k+1})$ for BLK_k as:

$$\bigwedge_{i=1}^{n_{k+1}} \text{SQ}(\mathbf{l}, D_i) \wedge \bigwedge_{i=1}^{n_{k+1}} \left(r_{i_{(n_k, D_i)}} \Leftrightarrow v_i^{(b)} \right),$$

where the construction of the vector $\mathbf{l}_k$ from the input vector $\mathbf{x}_k$ and the construction of the output vector $\mathbf{x}_{k+1}$ from $v_i^{(b)}$'s is as defined above.

Example 5. *Consider the internal block from Example 3. Following the above transformation, we can replace $x_{k_1} - x_{k_2} \geq -1 \Leftrightarrow v_1^{(b)} = 1$ as follows: $2x_{k_1}^{(b)} - 1 - (2x_{k_2}^{(b)} - 1) \geq -1 \Leftrightarrow v_1^{(b)} = 1$. Then we get $x_{k_1}^{(b)} - x_{k_2}^{(b)} \geq -0.5 \Leftrightarrow v_1^{(b)} = 1$. This constraint can be reexpressed as $x_{k_1}^{(b)} + \bar{x}_{k_2}^{(b)} \geq \lceil 0.5 \rceil \Leftrightarrow v_1^{(b)} = 1$, which can be encoded using sequential counters.*

Encoding of O**.** Consider the constraint in (8). We observe that the same transformation that we developed for the internal block can be applied here too. We perform variable substitutions, $x_{d_p} = 2x_{d_p}^{(b)} - 1$, where $x_{d_p}^{(b)} \in \{0, 1\}$ and

$p \in \{1, \ldots, n_d\}$. Let $E_{ij} = \lceil (b_j - b_i + \sum_{p=1}^{n_d} a_{ip} - \sum_{p=1}^{n_d} a_{jp})/2 \rceil$. We can reexpress (8) as:

$$\sum_{p \in \mathbf{a}_i^+} x_{d_p}^{(b)} - \sum_{p \in \mathbf{a}_i^-} x_{d_p}^{(b)} - \sum_{p \in \mathbf{a}_j^+} x_{d_p}^{(b)} + \sum_{p \in \mathbf{a}_j^-} x_{d_p}^{(b)} \geq E_{ij},$$

where $\mathbf{a}_i^+ = \{p \mid a_{ip} = 1\}$ and $\mathbf{a}_i^- = \{p \mid a_{ip} = -1\}$, and similarly $\mathbf{a}_j^+$ and $\mathbf{a}_j^-$. We can simplify this equation where each literal either occurs twice or cancels out as:

$$\sum_{p \in \mathbf{a}_i^+ \cap \mathbf{a}_j^-} x_{d_p}^{(b)} - \sum_{p \in \mathbf{a}_i^- \cap \mathbf{a}_j^+} x_{d_p}^{(b)} \geq \lceil E_{ij}/2 \rceil.$$

The rest of the encoding of these constraints is similar to BLK using sequential counters. This finishes the construction of $\text{BINO}(\mathbf{x}_d, o)$.

Example 6. *Consider the output block from Example 4. Following the above transformation, we can replace* $x_{d_1} - x_{d_2} \geq \lceil 0.7/2 \rceil \Leftrightarrow d_{12} = 1$ *as follows:* $2x_{d_1}^{(b)} - 1 - (2x_{d_2}^{(b)} - 1) \geq 1 \Leftrightarrow d_{12} = 1$. *Then we get* $x_{d_1}^{(b)} - x_{d_2}^{(b)} \geq 0.5 \Leftrightarrow d_{12} = 1$. *Finally, we rewrite as* $x_{d_1}^{(b)} + \bar{x}_{d_2}^{(b)} \geq \lceil 1.5 \rceil \Leftrightarrow d_{12} = 1$. *and can use sequential counters to encode the above expression.*

5 Encoding of Properties

In this section, we use the encoding constructed in the previous section to investigate properties of BNNs defined in Section 5. Note that since our encoding is exact, it allows us to investigate properties of BNNs in the SAT domain.

Verifying Adversarial Robustness. We need to encode the norm restriction on the perturbation and the adversarial condition. Consider an image $\mathbf{x} = (x_1, \ldots, x_n)$ and a perturbed image $\mathbf{x} + \tau$, where $\tau = (\tau_1, \ldots, \tau_n)$. Our encoding scheme works for any norm constraint that is a linear function of the input, which include the L_1- and the L_∞-norms. As discussed in Section 5, the most common norm assumption on τ is that of L_∞-norm, that we focus on here.

Norm Constraint: For $\|\tau\|_\infty \leq \epsilon$, we need to ensure that $\tau_i \in [-\epsilon, \epsilon]$ for all $i \in [n]$, where n is the size of the images. Note that τ_i is an integer variable as a valid image has integer values. Therefore, we can use the standard CNF conversion from linear constraints over integers into Boolean variables and clauses (Tamura et al. 2009). Additionally, we add a constraint to ensure that $\mathbf{x} + \tau$ is a valid image. For this, we make a natural assumption that exists a lower bound LB and an upper bound UB such that all the entries in a valid image lie within $[\text{LB}, \text{UB}]$. Then we add a constraint to ensure that all the entries in $\mathbf{x} + \tau$ lie within $[\text{LB}, \text{UB}]$.

Adversarial Constraint: We recall that an encoding of a BNN into SAT contains an integer output variable o. The value of o is the predicted label. Hence, we just need to encode that $o \neq \ell_\mathbf{x}$ into a Boolean formula, where $\ell_\mathbf{x}$ is the true label of $\mathbf{x}$. We use $\text{CNF}(\bullet)$ to denote a standard conversion of a constraint over integers into a Boolean formula. Putting these together, checking robustness translates into checking assignments for a formula $\text{BNN}_{Ad}(\mathbf{x} + \tau, o, \ell_\mathbf{x})$ defined as:

$$\text{BNN}_{Ad}(\mathbf{x}+\tau, o, \ell_\mathbf{x}) = \text{CNF}(\|\tau\|_\infty \leq \epsilon) \wedge \text{BNN}(\mathbf{x}+\tau, o) \wedge_{i=1}^{n} \text{CNF}((\mathbf{x}_i + \tau_i) \in [\text{LB}, \text{UB}]) \wedge \text{CNF}(o \neq l_X). \quad (13)$$

We can further simplify this equation by noting that for verifying the adversarial robustness property, we do not need to sort all outputs, but only need to ensure that $l_\mathbf{x}$ is not the top label in the ordering.

Verifying Universal Adversarial Robustness. This is now a simple extension to (13). For each $\mathbf{x}_i \in S$, we create a copy of the adversarial robustness property encoding $\text{BNN}_{Ad}(\mathbf{x}_i + \tau, o_i, \ell_{\mathbf{x}_i})$, and verify if at least ρ-fraction of the inputs are misclassified in S. This can be expressed as:

$$\bigwedge_{i=1}^{|S|} (\text{BNN}_{Ad}(\mathbf{x}_i + \tau, o_i, \ell_{\mathbf{x}_i}) \Leftrightarrow q_j) \wedge \text{CNF}\left(\sum_{i=1}^{|S|} q_j \geq \rho |S|\right).$$

Verifying Network Equivalence. To check the equivalence between two networks BNN_1 and BNN_2 we need to verify that these networks produce the same outputs on all valid inputs $\mathbf{x}$ in the domain. Again assuming that each entry in a valid image lies in the range $[\text{LB}, \text{UB}]$, we can formulate the network equivalence problem as the following decision problem on variables $\mathbf{x} = (x_1, \ldots, x_n)$.

$$\wedge_{i=1}^{n} \text{CNF}(x_i \in [\text{LB}, \text{UB}]) \wedge \text{BNN}_1(\mathbf{x}, o_1) \wedge \text{BNN}_2(\mathbf{x}, o_2) \wedge (o_1 \neq o_2). \quad (14)$$

If this formula is *unsatisfiable* then networks are equivalent. Otherwise, a solution of the formula is a valid witness on which these networks produce different outcomes.

6 Counterexample-Guided Search Procedure

Given the SAT formulas constructed in the previous section, we could directly run a SAT solver on them to verify the desired properties. However, the resulting encodings could be large with large networks, making them hard to tackle even for state-of-the-art SAT solvers. However, we can take advantage of the modular structure of the network to speed-up the search procedure. In particular, we use the idea of counterexample-guided search that is extensively used in formal verification (Clarke et al. 2003).

Observe that the SAT encoding follows the modular structure of the network. Let us illustrate our approach with a simple network consisting of two internal blocks and an output block as in Figure 1.[4] Suppose we want to verify the adversarial robustness property of the network (other properties can be handled similarly). The network can be encoded as a conjunction of two Boolean formulas: Gen (generator) that encodes the first block of the network, and Ver (verifier) that encodes the rest of the network. Therefore,

$$\text{BNN}_{Ad}(\mathbf{x} + \tau, o, \ell_\mathbf{x}) = \text{Gen}(\mathbf{x} + \tau, \mathbf{y}) \wedge \text{Ver}(\mathbf{y}, \mathbf{z}, o, \ell_\mathbf{x}),$$
$$\text{where } \text{Gen}(\mathbf{x} + \tau, \mathbf{y}) = \text{CNF}(\|\tau\|_\infty \leq \epsilon) \wedge$$
$$\wedge_{i=1}^{n} \text{CNF}((\mathbf{x}_i + \tau_i) \in [\text{LB}, \text{UB}] \wedge \text{BINBLK}_1(\mathbf{x} + \tau, \mathbf{y});$$
$$\text{Ver}(\mathbf{y}, \mathbf{z}, o, \ell_\mathbf{x}) = \text{BINBLK}_2(\mathbf{y}, \mathbf{z}) \wedge \text{BINO}(\mathbf{z}, o) \wedge \text{CNF}(o \neq l_X).$$

The generator and the verifier only share variables in $\mathbf{y}$ that encode activations shared between Block 1 and Block 2 of the

[4] The approach easily extends to a general network were the partitioning can be applied after each block of layer.

network. The set of variables $\mathbf{y}$ is usually small compared to all variables in the formula. We exploit this property by using *Craig interpolants* to build an efficient search procedure.

Definition 5 (Craig Interpolants)**.** *Let A and B be Boolean formulas with some shared variables such that the formula $A \wedge B$ is unsatisfiable. Then there exists a formula I, called interpolant, such that* $\text{vars}(I) = \text{vars}(A) \cap \text{vars}(B)$*, $B \wedge I$ is unsatisfiable and $A \Rightarrow I$. In general, there exist multiple interpolants for the given A and B.*

An interpolant can be obtained from a proof of unsatisfiability of $A \wedge B$ produced by a SAT solver (**?**).

Our search procedure first generates a satisfying assignment to variables τ and $\mathbf{y}$ for the generator formula $\text{Gen}(\mathbf{x} + \tau, \mathbf{y})$. Let $\hat{\mathbf{y}}$ denote this assignment to $\mathbf{y}$. Then we check if we can extend the assignment $\mathbf{y} = \hat{\mathbf{y}}$ to a satisfying assignment for the verifier formula. If so, we found an adversarial perturbation τ. Otherwise, we generate an interpolant I of $\text{Gen}(\mathbf{x} + \tau, \mathbf{y}) \wedge \text{Ver}(\mathbf{y} = \hat{\mathbf{y}}, \mathbf{z}, o, \ell_{\mathbf{x}})$ by extracting an unsatisfiable core of $\text{Ver}(\mathbf{y} = \hat{\mathbf{y}}, \mathbf{z}, o, \ell_{\mathbf{x}})$ (Biere et al. 2009). Since none of the satisfying assignments to I can be extended to a valid satisfying assignment of $\text{BNN}_{Ad}(\mathbf{x} + \tau, o, \ell_{\mathbf{x}})$, we block them all in Gen by redefining: $\text{Gen} := \text{Gen} \wedge \neg I$. Then we repeat the procedure. As we are reducing the solution space in each iteration, this algorithm terminates. If the formula $\text{Gen}(\mathbf{x} + \tau, \mathbf{y})$ becomes unsatisfiable, then there is no valid perturbation τ, i.e., the network is ϵ-robust on image $\mathbf{x}$.

7 Related Work

With the wide spread success of deep neural network models, a new line of research on understanding neural networks has emerged that investigate a wide range of these questions, from interpretability of deep neural networks to verifying their properties (Pulina and Tacchella 2010; Bastani et al. 2016; Huang et al. 2017; Katz et al. 2017).

(Pulina and Tacchella 2010) encode a neural network with sigmoid activation functions as a Boolean formula over linear constraints which has the same spirit as our MILP encoding. They use piecewise linear functions to approximate non-linear activations. The main issue with their approach is that of scalability as they can only verify properties on small networks, e.g., 20 hidden neurons. Recently, (Bastani et al. 2016) propose to use an LP solver to verify properties of a neural network. In particular, they show that the robustness to adversarial perturbations can be encoded as a system of constraints. For computational feasibility, the authors propose an approximation of this constraint system to a linear program. Our approach on the other hand is an exact encoding that can be used to verify any property of a binarized neural network, not just robustness. (Katz et al. 2017) consider neural networks with ReLU activation functions and show that the Simplex algorithm can be used to deal with them. The important property of this method is that it works with a system of constraints directly rather than its approximation. However, the method is tailored to the ReLU activation function. Scalability is also a concern in this work as each ReLU introduces a branching choice (an SMT solver is used to handle branching). The authors work with networks with 300 ReLUs in total and 7 inputs. Finally, (Huang et al. 2017) perform discretization of real-valued inputs and use a counterexample-guided abstraction refinement procedure to search for adversarial perturbations. The authors assume an existence of inverse functions between layers to perform the abstraction refinement step. Our method also utilizes a counterexample-guided search. However, as we consider blocks of layers rather than individual layers we do not need an abstraction refinement step. Also we use interpolants to guide the search procedure rather than inverse functions.

Finally, to the best of our knowledge, the MILP encoding is the only *complete* search procedure for finding adversarial perturbations previously proposed in the literature. In our experiments, as a baseline, we use the ILP encoding (described in Section 4), as it is a more efficient encoding than MILP for BNNs. Most of the other existing adversarial perturbation approaches are *incomplete* search procedures, and are either greedy-based, e.g., the fast gradient method (Goodfellow, Shlens, and Szegedy 2015), or tailored to a particular class of non-linearity, e.g., ReLU (Katz et al. 2017).

8 Experimental Evaluation

We used the Torch machine learning framework to train networks on a Titan Pascal X GPU. We trained networks on the MNIST dataset (LeCun et al. 1998), and two modern variants of MNIST: the MNIST-rot and the MNIST-back-image (Larochelle, Erhan, and Courville 2017). In the MNIST-rot variation of the dataset, the digits were rotated by an angle generated uniformly between 0 and 2π radians. In the MNIST-back-image variation of the dataset, a patch from a black-and-white image was used as the background for the digit image. In our experiments, we focused on the important problem of checking adversarial robustness under the L_∞-norm.

We next describe our BNN architecture. Our network consists of four internal blocks with each block containing a linear layer (LIN) and a final output block. The linear layer in the first block contains 200 neurons and the linear layers in other blocks contain 100 neurons. We use the batch normalization (BN) and binarization (BIN) layers in each block as detailed in Section 3. Also as mentioned earlier, there is an additional hard tanh layer in each internal block that is only used during training. To process the inputs, we add two layers (BN and BIN) to the BNN, as the first two layers in the network to perform binarization of the gray-scale inputs. This simplifies the network architecture and reduces the search space. This addition decreases the accuracy of the original BNN network by less than one percent. The accuracy of the resulting network on the MNIST, MNIST-rot, and MNIST-back-image datasets were 95.7%, 71% and 70%, respectively.

To check for adversarial robustness, for each of the three datasets, we randomly picked 20 images (from the test set) that were correctly classified by the network for each of the 10 classes. This resulted in a set of 200 images for each dataset that we consider for the remaining experiments. To reduce the search space we first focus on important pixels of an image as defined by the notion of *saliency map* (Simonyan, Vedaldi, and Zisserman 2014). In particular, we first try to

	Solved instances (out of 200)									Certifiably ϵ-robust		
	MNIST			MNIST-rot			MNIST-back-image					
	SAT	ILP	CEG	SAT	ILP	CEG	SAT	ILP	CEG	SAT	ILP	CEG
	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#solved (t)	#	#	#
$\epsilon = 1$	180 (77.3)	130 (31.5)	171 (34.1)	179 (57.4)	125 (10.9)	197 (13.5)	191 (18.3)	143 (40.8)	191 (12.8)	138	96	138
$\epsilon = 3$	187 (77.6)	148 (29.0)	181 (35.1)	193 (61.5)	155 (9.3)	198 (13.7)	107 (43.8)	67 (52.7)	119 (44.6)	20	5	21
$\epsilon = 5$	191 (79.5)	165 (29.1)	188 (36.3)	196(62.7)	170(11.3)	198(13.7)	104 (48.8)	70 (53.8)	116 (47.4)	3	–	4

Table 2: Results on MNIST, MNIST-rot and MNIST-back-image datasets.

perturb the top 50% of highly salient pixels in an image. If we cannot find a valid perturbation that leads to misclassification among this set of pixels then we search again over all pixels of the image. We experimented with three different maximum perturbation values by varying $\epsilon \in \{1, 3, 5\}$. The timeout is 300 seconds for each instance to solve.

We compare three methods of searching for adversarial perturbations. The first method is an ILP method where we used the SCIP solver (Maher et al. 2017) on the ILP encoding of the problem (denoted ILP). The second method is a pure SAT method, based on our proposed encoding, where we used the Glucose SAT solver (Audemard and Simon 2009) to solve the resulting encoding (denoted SAT). For the third method, we used the same SAT encoding but for speeding-up the search we utilized the counterexample-guided search procedure described in Section 6. We use the core produced by the verifier as an interpolant. We call this method CEG. On average, our generated SAT formulas contain about 1.4 million variables and 5 million clauses. The largest instance contains about 3 million variables and 12 million clauses.

Table 2 presents the results for different datasets. In each column we show the number of instances solved by the corresponding method (#solved) out of the 200 selected instances and the average time in seconds (t) to solve these instances (within the 300 seconds timeout). Solved in this context means that either we determine a valid perturbation leading to misclassification of the image by the network, or concretely establish that there exists no solution which means that the network is ϵ-robust on this image.

It is clear that SAT and CEG methods outperform the ILP approach. The ILP method solves up to 30% fewer instances compared to SAT or CEG across all datasets. This demonstrate the effectiveness of our SAT encoding compared to the ILP encoding. Comparing the SAT and CEG methods, we see that the results are mixed. On the MNIST-rot and MNIST-back-image datasets CEG solves more instances than SAT, while on the MNIST dataset we have the opposite situation. We observe that CEG is faster compared to SAT across all datasets. Generally, we noticed that the images in the MNIST-rot dataset were easiest to perturb, whereas images in the MNIST-back-image dataset were the hardest to perturb.

A major advantage of our complete search procedure is that we can *certify* ϵ-robustness, in that there exists no adversarial perturbation technique that can fool the network on these images. Such guarantees cannot be provided by incomplete adversarial search algorithms previously considered in the literature (Szegedy et al. 2014; Goodfellow, Shlens, and Szegedy 2015; Moosavi-Dezfooli, Fawzi, and Frossard 2016). In the last three columns of Table 2, we present the number of images in the MNIST-back-image dataset on which the network is certifiably ϵ-robust as found by each of the three methods. Again the SAT-based approaches outperform the ILP approach. With increasing ϵ, the number of images on which the network is ϵ-robust decreases as the adversary can leverage the larger ϵ value to construct adversarial images. This decrease is also reflected in Table 2 (Columns 7–9) in terms of the number of solved instances which decreases from $\epsilon = 3$ to $\epsilon = 1$, as some of the instances on which a lack of solution can be certified at $\epsilon = 1$ by a method cannot always be accomplished at $\epsilon = 3$ within the given time limit.

Examples of Perturbed Images. Table 3 shows examples of original and successfully perturbed images generated by our SAT method. We have two images from each dataset in the table. As can be seen from these pictures, the differences are sometime so small that original and perturbed images, that they are indistinguishable for the human eye. These examples illustrate that it is essential that we use a certifiable search procedure to ensure that a network is robust against adversarial inputs.

9 Conclusion

We proposed an exact Boolean encoding of binarized neural networks that allows us to verify interesting properties of these networks such as robustness and equivalence. We further proposed a counterexample-guided search procedure that leverages the modular structure of the underlying network to speed-up the property verification. Our experiments demonstrated feasibility of our approach on the MNIST and its variant datasets. Our future work will focus on improving the scalability of the proposed approach to enable property verification of even larger neural networks by exploiting the structure of the formula. Another interesting direction is to verify properties of recurrent neural networks that have a number of additional structural properties, e.g., repetitive blocks of layers, that can be exploited during reformulation and search.

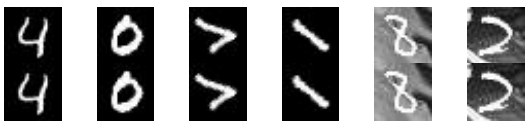

Table 3: The original (the top row) and successfully perturbed (the bottom row) images.

References

[Audemard and Simon 2009] Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern sat solvers. In *IJCAI*, volume 9, 399–404.

[Bastani et al. 2016] Bastani, O.; Ioannou, Y.; Lampropoulos, L.; Vytiniotis, D.; Nori, A.; and Criminisi, A. 2016. Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems*, 2613–2621.

[Bau et al. 2017] Bau, D.; Zhou, B.; Khosla, A.; Oliva, A.; and Torralba, A. 2017. Network Dissection: Quantifying Interpretability of Deep Visual Representations. *CoRR* abs/1704.0.

[Biere et al. 2009] Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. {IOS} Press.

[Clarke et al. 2000] Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Computer aided verification*, 154–169. Springer.

[Clarke et al. 2003] Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM* 50(5):752–794.

[Goodfellow, Shlens, and Szegedy 2015] Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In *ICLR*.

[Griva, Nash, and Sofer 2009] Griva, I.; Nash, S. S. G.; and Sofer, A. 2009. *Linear and nonlinear optimization*. Society for Industrial and Applied Mathematics.

[Grosse et al. 2016] Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; and McDaniel, P. 2016. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*.

[Huang et al. 2017] Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In *{CAV} {(1)}*, volume 10426 of *Lecture Notes in Computer Science*, 3–29. Springer.

[Hubara et al. 2016] Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized Neural Networks. In *Advances in Neural Information Processing Systems 29*, 4107–4115.

[Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, 448—-456. JMLR.org.

[Katz et al. 2017] Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*.

[Kung et al. 2017] Kung, J.; Zhang, D.; van der Wal, G.; Chai, S.; and Mukhopadhyay, S. 2017. Efficient object detection using embedded binarized neural networks. *Journal of Signal Processing Systems* 1–14.

[Larochelle, Erhan, and Courville 2017] Larochelle, H.; Erhan, D.; and Courville, A. 2017. MnistVariations < Public < TWiki.

[LeCun et al. 1998] LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

[Maher et al. 2017] Maher, S. J.; Fischer, T.; Gally, T.; Gamrath, G.; Gleixner, A.; Gottwald, R. L.; Hendel, G.; Koch, T.; Lübbecke, M. E.; Miltenberger, M.; Müller, B.; Pfetsch, M. E.; Puchert, C.; Rehfeldt, D.; Schenker, S.; Schwarz, R.; Serrano, F.; Shinano, Y.; Weninger, D.; Witt, J. T.; and Witzig, J. 2017. The scip optimization suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin.

[McDanel, Teerapittayanon, and Kung 2017] McDanel, B.; Teerapittayanon, S.; and Kung, H. 2017. Embedded binarized neural networks. In *International Conference on Embedded Wireless Systems and Networks*.

[Moosavi-Dezfooli et al. 2016] Moosavi-Dezfooli, S.-M.; Fawzi, A.; Fawzi, O.; and Frossard, P. 2016. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*.

[Moosavi-Dezfooli, Fawzi, and Frossard 2016] Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. DeepFool: a simple and accurate method to fool deep neural networks. In *CVPR*.

[Papernot et al. 2016] Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 372–387. IEEE.

[Pulina and Tacchella 2010] Pulina, L., and Tacchella, A. 2010. An abstraction-refinement approach to verification of artificial neural networks. In *Computer Aided Verification*, 243–257. Springer.

[Pulina and Tacchella 2012] Pulina, L., and Tacchella, A. 2012. Challenging SMT solvers to verify neural networks. *AI Communications* 25(2):117–135.

[Reagen et al. 2017] Reagen, B.; Adolf, R.; Whatmough, P.; Wei, G.-Y.; and Brooks, D. 2017. Deep Learning for Computer Architects. *Synthesis Lectures on Computer Architecture* 12(4):1–123.

[Simonyan, Vedaldi, and Zisserman 2014] Simonyan, K.; Vedaldi, A.; and Zisserman, A. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR Workshop*.

[Sinz 2005] Sinz, C. 2005. Towards an optimal cnf encoding of boolean cardinality constraints. *CP* 3709:827–831.

[Springenberg et al. 2015] Springenberg, J.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2015. Striving for Simplicity: The All Convolutional Net. In *ICLR (workshop track)*.

[Szegedy et al. 2014] Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *ICLR*.

[Tamura et al. 2009] Tamura, N.; Taga, A.; Kitagawa, S.; and Banbara, M. 2009. Compiling finite linear csp into sat. *Constraints* 14(2):254–272.