How Many Bits Does it Take to Quantize Your Neural Network?

Mirco Giacobbe, Thomas A. Henzinger and Mathias Lechner

May 25, 2019

How Many Bits Does it Take to Quantize Your Neural Network?

Mirco Giacobbe, Thomas A. Henzinger, and Mathias Lechner
IST Austria

***Abstract*—Quantization converts neural networks from traditional floating-point to low-bit fixed-point computations, which can be carried out by efficient integer-only hardware. Quantization is standard practice for the deployment of neural networks on real-time embedded devices, as quantized networks use little memory and yield low latency for classifying inputs. However, like their real-number counterpart, quantized networks are not immune to malicious misclassifications caused by adversarial attacks. Our experiments show that the analysis of real-numbered networks often derives false conclusions about their quantizations, both when determining robustness and when detecting attacks. For this reason, we investigate how quantization affects a network's robustness to adversarial attacks, which is a formal verification question. We introduce a verification method for quantized neural networks which, using SMT solving over bit-vectors, accounts for their exact, bit-precise semantics. We built a tool and investigated the effect of quantization for multiple case studies. First, we analyzed a classifier for the MNIST dataset, answering the question as to how many bits are necessary for quantization to achieve robustness to attacks. Second, we examined a neural controller for the stabilization of an inverted pendulum, measuring the behavioral differences between different precisions used in quantizations. Finally, we verified the gender bias of a student performance predictor, assessing how many bits are necessary in quantization for safety properties to be satisfied.**

I. Introduction

Deep Neural Networks are the standard among all machine learning models, and are nowadays present in almost every software we use. In the recent years, they have also pervaded our lives: think about the language recognition system of a voice assistant, the computer vision employed in face recognition or self driving, not to talk about many decision-making tasks that are hidden under the hood. However, this also subjects them to the resource limits that real-time embedded devices impose. Mainly, the requirements are low energy consumption, as they often run on batteries, and low latency, both to maintain user engagement and to effectively interact with the physical world. This translates into specializing our computation by reducing memory footprint and set of instructions, to minimize cache misses avoid costly hardware operations. For this purpose, quantization compresses traditional neural networks, which are run over 32- or 64-bit floating-point arithmetic, into computations that only require bit-wise and integer-only arithmetic over small words, e.g., 8 bits. Quantization is the standard technique for the deployment of neural networks on mobile and embedded devices, and is implemented in TensorFlow Lite [1], [2]. In this work, we investigate the robustness of quantized networks to adversarial attacks and, more generally, to formal verification questions, including equivalence and safety.

Adversarial attacks are a well-known vulnerability of neural networks [3]. For instance, a self-driving car can be tricked into confusing a stop with a speed limit sign [4], or a home automation system can be commanded to deactivate the security camera by a voice reciting the opening of the Iliad [5]. The attack is carried out by superposing the innocuous input with a crafted perturbation that is imperceptible to humans. Formally, the attack lies within the neighborhood of a known-to-be-innocuous input, according to some notion of distance. The fraction of samples (from a large set of test inputs) that do not admit attacks determines the robustness of the network. We ask ourselves how quantization affects networks' robustness or, dually, how many bits it takes to keep robustness above some specific threshold. This amounts to proving that, for a set of given quantizations and inputs, there does not exists an attack, which is a formal verification question.

The formal verification of neural networks has been addressed either by overapproximating—as it happens in abstract interpretation—the space of outputs given a space of attacks, or by searching—as it happens in SMT-solving—for a variable assignment that witnesses an attack. The first category include methods that relax the neural networks into computations over interval arithmetic [6], treat them as hybrid automata [7], or abstract them directly by using zonotopes, polyhedra [8], or tailored abstract domains [9]. Overapproximation-based methods are typically fast, but incomplete: they prove robustness but do not produce attacks. On the other hand, methods based on local gradient decent have turned out to be effective in producing attacks in many cases [10], but sacrifice formal completeness. Indeed, the search for adversarial attack is NP-complete even for the simplest (i.e., ReLU) networks [11], which motivates the rise of methods based on *Satisfiability Modulo Theory* (SMT) and *Mixed Integer Linear Programming* (MILP). SMT-solvers have been shown not to scale beyond toy examples (20 hidden neurons) on monolithic encodings [12], but today's specialized techniques can handle real-life benchmarks such as, e.g., neural networks for the MNIST dataset [13]. Specialized tools include DLV [14], which subdivides the problem into smaller SMT instances, and Planet [15], which combines different SAT and LP relaxations. Reluplex takes a step further augmenting LP-solving with a custom calculus for ReLU networks [11]. On the other side of the spectrum, a recent MILP formulation turned out effective using off-the-shelf solvers [16]. Moreover, it posed the basis for Sherlock [17], which couples local search and MILP, and for a specialized branch and bound algorithm [18].

All techniques mentioned above reason about the real-number relaxation of the network. While adversarial attacks for the reals are likely to be also attacks in practice, namely on floating-point architectures, it follows from our experiments that this is not the case for quantized neural networks. In fact, we observed that verifying the real-relaxation may (i) conclude that samples are robust while they admit attacks under quantization (false negative), but also may (ii) find attacks for samples that are instead robust under quantization (false positive). In addition, it may (iii) correctly identify samples as vulnerable but provide invalid attacks, and all three phenomena happen with statistical significance on our benchmarks. For this reason, the verification of real-numbered neural networks is inadequate for the analysis of quantized networks, and their analysis needs techniques that account for their exact semantics. Recently, a similar problem has been addressed on binarized neural networks, through SAT-solving [19]. Unfortunately, binarized networks amount to the special case of 1-bit quantizations, therefore the method is unsuitable for any many-bit quantization in TensorFlow Lite.

We introduce, for the first time, a method for the formal verification of quantized neural networks. Our method accounts for the bit-precise semantics of quantized networks by leveraging the first-order theory of bit vectors without quantifiers (QF_BV) to exactly encode hardware operations such as 2'complementation, bit-shift, integer arithmetic with overflow. On the technical side, we encode multiply-add operations in a balanced fashion, which enabled the SMT-solver to scale up to our benchmarks. As a result, we obtain a monolithic encoding of the verification problem into a first-order logic formula, amenable to modern bit-precise SMT-solving. We built a tool based on the SMT-solver Boolector [20], which we used to assess the effect of quantization for multiple networks and verification questions.

First, we measured the robustness to attacks of multiple quantizations of a neural network trained as a classifier for handwritten digits, after the MNIST dataset. We observed that, to achieve an acceptable level of robustness, it takes a higher bit quantization than assessed by standard accuracy measures. Besides, this experiment employed our largest benchmarks. On one hand, it demonstrated the potential of our method which, using an SMT-solver without custom optimizations, could tackle 6- to 10-bits quantizations of a network involving 890 neurons; on the other, it showed the limits of current solvers, with a median instance of 3h 41m and a hardest of over 12 hours.

Second, we measured the behavioral equivalence between various precisions in the quantization of a neural controller for an inverted pendulum, trained by reinforcement learning. In particular, we measured the discrepancy between the outputs of every pair of networks, quantized using a 6- to 10-bits precision. We observed that down- but also up-quantization may have an effect upon the behavior of a neural controller.

Third, we checked the robustness of safety properties against quantization. We estimated the gender bias emerging from a predictor of student performance in exams, defined in terms of maximum grade gap between any two student with identical features but the gender. The experiment confirmed that, in our network, a bias existed and was further enlarged by quantization: the lower the precision the larger the gap.

We summarize our contribution in four points. First, we show that the robustness of a neural network is independent of the robustness of its quantizations. Second, we introduce the first method for the verification of quantized neural networks. Third, we build a tool and demonstrate that, for instances with hundreds of neurons, quantized networks can be verified using SMT-solving. Fourth, we show that quantization has an effect upon the robustness of neural networks, not only with respect to adversarial attack, but also with respect to general verification questions, such as equivalence and safety.

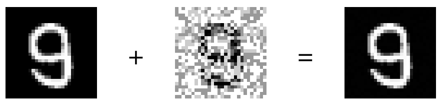


Figure 1: Adversarial attack.[1]

II. Real and Quantized Networks are Incomparable

A classifier for handwritten digits takes an image and infers the digit it represents. Then, an adversarial attack is a perturbations for a sample image

$$\text{sample} + \text{perturbation} = \text{attack}$$

that is indistinguishable from the original by the human eye, but tricks the classifier into inferring an incorrect digit. An attack commonly consists of the modification of a few pixels and of small amounts, such as Fig. 1, which depicts an attack for a neural network that realizes a classifier for the MNIST library. In this particular example, the sample is correctly classified as a 9, while the attack is misclassified as a 3. Misclassification happens consistently, both on the floating-point and on the 8-bits quantized neural network. This makes a valid attack in both networks which, unfortunately, is not always the case.

We want to estimate whether proving or rejecting the robustness over the real relaxation concludes the same result about the quantized network. To this end, we chose 244 test samples from the dataset. Then, using our tool, we determined over the quantized network whether each sample admits an adversarial perturbation, which makes it *vulnerable*, or whether it does not, which makes it *robust*. Finally, we used Reluplex to do the same over the real-number relaxation.

Our experiments reported several spurious outcomes, which we exemplify in Tab. I: for three test samples, we show verification outcomes for the real-numbered and the 8-bit quantized networks, attacks, and inference outcomes for the floating-point and the 8-bit fixed-point implementations. In

[1]In all figures, perturbations are amplified to facilitate visualization.

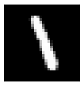
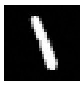
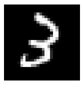
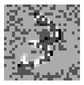
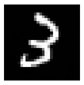
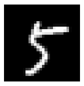
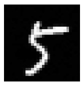
Vulnerable		Attack					Inference		
ℝ	8-bits						float	8-bits	
No	Yes		+		=		1	**7**	(i)
Yes	No		+		=		**2**	3	(ii)
Yes	Yes		+		=		**7**	5	(iii)

Table I: Spurious verification outcomes.

particular, these are examples where the real-number network (i) determined samples robust while attacks existed in the quantization, (ii) proved the existence of attacks for samples that did not admit one, and (iii) concluded samples vulnerable but identifying invalid attacks. Notably, none of these examples were isolated cases: Tab. II shows that spurious outcomes are not rare, especially at lower precisions. Moreover, while false outcomes decrease with precision, the ratio of correct positives that are invalid attacks, shown in the the rightmost column, is significant through all quantizations.

		(i)	(ii)		(iii)
Bits	Correct negatives	False negatives	False positives	Correct positives	Invalid attacks
6	66.4%	25.0%	3.3%	5.3%	8%
7	84.8%	6.6%	1.6%	7.0%	6%
8	88.5%	2.9%	0.4%	8.2%	10%
9	91.0%	0.4%	0.4%	8.2%	20%
10	91.0%	0.4%	0.4%	8.2%	20%

Table II: Frequency of verification outcomes.

Not only attacks can be invalid, but can also be non-monotonic. In fact, as we illustrate in Tab. III, attacks that induce misclassification to specific quantizations do not necessarily induce misclassification using lower, neither higher, precision.

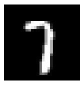
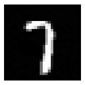
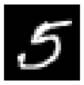
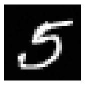
Attack					Inference			
					float	9-bits	8-bits	7-bits
	+		=		3	7	3	3
	+		=		0	5	0	5

Table III: Non-monotonic attacks.

In conclusion, verifying a quantized neural network is incomparable to verifying its relaxation over the real numbers; also, it is incomparable to verifying the same network under different quantization precisions. For this reason, we introduce a method for the bit-precise verification of quantized networks: in Sec. III we give the preliminaries about neural networks and their quantization, and in Sec. IV we present a translation of the verification problem into SMT. Then, in Sec. V, we give our experimental results.

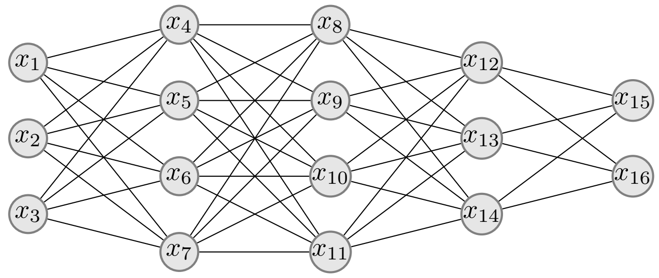


Figure 2: A feed-forward neural network.

III. Quantization of Feed-forward Networks

A feed-forward neural network consists of a finite set of *neurons* $x_1, \ldots, x_k$ partitioned into a sequence of *layers*: an *input* layer with n neurons, followed by one or many *hidden* layers, finally followed by an *output* layer with m neurons. For instance, Fig. 2 depicts a neural network with $k = 16$, $n = 3$ input neurons (x_1, x_2, x_3), $m = 2$ output neurons (x_{15}, x_{16}), and 11 hidden neurons partitioned in 3 layers, each of which contains respectively 4, 4, and 3 neurons. Every pair of neurons x_j and x_i in respectively subsequent layers, e.g., x_1 and x_4, are connected and associated with a *weight* coefficient $w_{ij} \in \mathbb{R}$ (if they are not subsequent, e.g., x_1 and x_8, then $w_{ij} = 0$) and every hidden or output neuron x_i is associated with a *bias* coefficient $b_i \in \mathbb{R}$. The semantics of the neural network gives to each neuron a real value: upon a valuation for the input layer, every other neuron x_i assumes its value according to the update rule

$$x_i = \mathrm{ReLU}\text{-}N(b_i + \sum_{j=1}^{k} w_{ij} x_j), \tag{1}$$

where $\mathrm{ReLU}\text{-}N \colon \mathbb{R} \to \mathbb{R}$ is the *activation function*. Altogether, the neural network implements a function $f \colon \mathbb{R}^n \to \mathbb{R}^m$ whose result corresponds to the valuation for the output layer.

The activation function governs the firing logic of the neurons, layer by layer, by introducing non-linearity in the system. Among the most popular activation functions are purely non-linear functions, such as the tangent hyperbolic and the sigmoidal function, and piece-wise linear functions, better known as *Rectified Linear Units* (ReLU) [21]. ReLU consists of the function that takes the positive part of its argument, i.e., $\mathrm{ReLU}(x) = \max\{x, 0\}$. We consider the variant of ReLU that imposes a cap value N, known as ReLU-N [22], [23]. Precisely

$$\mathrm{ReLU}\text{-}N(x) = \min\{\max\{x, 0\}, N\}, \tag{2}$$

which can be alternatively seen as a concatenation of two ReLU (see Eq. 11). As a consequence, our neural networks will be also amenable to the precise verification over linear real arithmetic, allowing the comparison of our method against the real-valued verifier implemented in Reluplex [11]. Nevertheless, our method in principle applies to purely non-linear functions too.

Quantization[2] consists of converting a neural network over real numbers, which is normally deployed on floating-point architectures, into a neural network over integers, whose semantics corresponds to a computation over fixed-point arithmetic [1], [2]. Specifically, fixed-point arithmetic can be carried out by integer-only architectures and possibly over small words, e.g., 8 bits. Quantization represents all numbers in 2's complement over B bits words and reserves F bits to the fractional part: we call the result a B-bits quantization in QF arithmetic. More concretely, the conversion follows from the rounding of weight and bias coefficients to the F-th digit, namely $\bar{b}_i = \lfloor 2^F b_i \rceil$ and $\bar{w}_{ij} = \lfloor 2^F w_{ij} \rceil$, where $\lfloor \cdot \rceil$ stands for rounding to the nearest integer. Therefore, the fundamental relation between any quantized value $\bar{a}$ and its real counterpart a is

$$a \approx 2^{-F} \bar{a}. \tag{3}$$

As a consequence, the semantics of a quantized neural network corresponds to the update rule in Eq. 1 after substituting of x, w, and b with the respective approximants $2^{-F}\bar{x}$, $2^{-F}\bar{w}$, and $2^{-F}\bar{b}$. Namely, the semantics amounts to

$$\bar{x}_i = \text{ReLU-}(2^F N)(\bar{b}_i + \lfloor 2^{-F} \sum_{j=1}^{k} \bar{w}_{ij}\bar{x}_j \rfloor), \tag{4}$$

where truncation $\lfloor \cdot \rfloor$ enforces $\bar{x}_i$ to represent the result in QF arithmetic. In summary the update rule for the quantized semantics consists of four parts. The first part $\sum_{j=1}^{k} \bar{w}_{ij}\bar{x}_j$ propagates all neurons values from the previous layer. The second scales the result by 2^{-F} truncating the fractional part; in practice, it applies an arithmetic shift to the right of F bits. Finally, the third applies the bias $\bar{b}$ and the fourth clamps the result between 0 and $2^F N$. Besides, none of the operations above incur in overflow if B is chosen properly. More precisely, this is the case when all words can represent (in 2's complement) the magnitude of weight and bias coefficients, i.e., $|\bar{w}| < 2^{B-1}$ and $|\bar{b}| < 2^{B-1}$, and of neurons, i.e., $\bar{x} \leq 2^F N < 2^{B-1}$. In conclusion, quantization realizes a function $f \colon \mathbb{Z}^n \to \mathbb{Z}^m$, whose evaluation relies on fixed-point operations only. It makes inference deployable on efficient integer-only hardware, but requires care in choosing the parameters (see Sec. IV and Sec. V).

In summary, the semantics of a quantized network involves integer-only hardware operations which, in their turn, have corresponding primitives in the SMT theory of bit-vectors. Thanks to that, in Sec. IV we encode the verification problem of quantized networks into SMT.

IV. Verification of Quantized Networks using Bit-precise SMT-solving

Bit-precise SMT-solving comprises various technologies for deciding the satisfiability of first-order logic formulae, whose variables are interpreted as bit-vectors of fixed size. In particular, it produces satisfying assignments (if any exist) for formulae that include bitwise and arithmetic operators, whose semantics corresponds to that of hardware architectures. For instance, we can encode bit-shifts, 2's complementation, multiplication and addition with overflow, signed and unsigned comparisons. More precisely, this is the quantifier-free first-order theory of bit-vectors (i.e., QF_BV), which we employ to produce a monolithic encoding of the verification problem for quantized neural networks.

A verification problem for the neural networks $f_1, \dots, f_K$ consists of checking the validity of a statement of the form

$$\varphi(\vec{y}_1, \dots, \vec{y}_K) \implies \psi(f_1(\vec{y}_1), \dots, f_K(\vec{y}_K)), \tag{5}$$

where φ is a predicate over the inputs and ψ over the outputs of all networks. In other words, it consists of checking an assume–guarantee contract, for the *assumption* φ and the *guarantee* ψ; this generalizes various verification questions, including robustness to adversarial attacks and all other questions we treat in Sec. V. For the purpose of SMT solving, we encode the verification problem in Eq. 5, which is a validity question, by its dual satisfiability question

$$\varphi(\vec{y}_1, \dots, \vec{y}_K) \wedge \bigwedge_{i=1}^{K} f_i(\vec{y}_i) = \vec{z}_i \;\wedge\; \neg\psi(\vec{z}_1, \dots, \vec{z}_K), \tag{6}$$

whose satisfying assignments constitute counterexamples for the contract. The formula consists of three conjuncts: the rightmost constraints the input within the assumption, the leftmost forces the output to violate the guarantee, while the one in the middle relates inputs and outputs by the semantics of the neural networks.

The semantics of the network consists of the bit-level translation of the update rule in Eq. 4 over all neurons, which we encode in the formula

$$\begin{aligned} \bigwedge_{i=1}^{k} x_i = \text{ReLU-}(2^F N)(x'_i) & \\ \wedge\; x'_i = \bar{b}_i + \texttt{ashr}(x''_i, F) & \\ \wedge\; x''_i = \sum_{j=1}^{k} \bar{w}_{ij} x_j. & \end{aligned} \tag{7}$$

Each conjunct in the formula employs three variables x, x', and x'' and is made of three parts. The highermost part accounts for the operation of clamping between 0 and $2^F N$, whose semantics is given by the formula

$$\text{ReLU-}M(x) = \texttt{ite}(\texttt{sign}(x), 0, \texttt{ite}(x \geq M, M, x)). \tag{8}$$

Then, the central part accounts for the operations of scaling and biasing. In particular, it encodes the operation of truncated scaling, i.e., $\lfloor 2^{-F} x \rfloor$, as an arithmetic shift to the right. Finally, the lowermost part accounts for the propagation of values from the previous layer, which, despite the obvious optimization of pruning away all monomials with null coefficient, often consists of long linear combinations, whose exact semantic amounts to a sequence of multiply-add operations over an accumulator; particularly, encoding it requires care in choosing variables size and association layout.

[2]Our quantization scheme is similar to the scheme in [2], for the parameters $Z = 0$ and $S = 2^{-F}$.

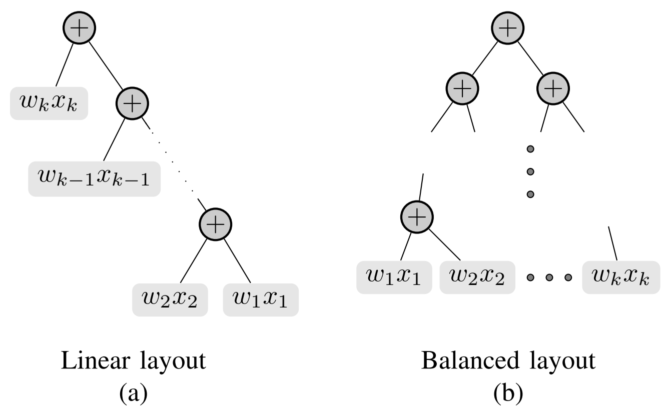


Figure 3: Abstract syntax trees for alternative encodings of a long linear combination.

The size of the bit-vector variables determines whether overflows can occur. In particular, since every monomial $w_{ij}x_j$ consists of the multiplication of two B-bits variables, its result requires $2B$ bits in the worst case; since summation increases the value linearly, its result requires a logarithmic amount of extra bits in the number of summands (regardless of the layout). Provided that, we avoid overflow by using variables of $2B + \log k$ bits, where k is the number of summands.

The association layout is not unique and, more precisely, varies with the order of construction of the long summation. For instance, consistently associating to the right (or to the left) produces a linear layout, as in Fig. 3a, while recursively splitting the sum in two equal parts produces a balanced layout, as in Fig. 3b. Linear and balanced layouts are semantically equivalent; nevertheless, the first made the solver, Boolector [20], timing-out on all benchmarks. Conversely, the balanced layout made the solver scale up, yielding the results of Sec. V. Besides, this suggests to us that the solver does not preprocess long linear combinations in the same way.

V. Experimental Results

We investigate the effect of quantization upon various verification questions and, at the same time, we evaluate the performance of our method. We verify three questions of the assume–guarantee kind (i.e., as in Eq. 5): robustness against adversarial attacks, equivalence, and finally general safety properties. As for robustness against adversarial attacks, we study how quantization statistically affects robustness over different precision levels in the quantization of a classifier for handwritten digits; also, we extend the analysis of Sec. II, comparing bit-precise against real-numbered verification. In more detail, we analyze a fully-connected network with 890 neurons in Sec. V-A, and, to take our method to its limit, we analyze a Convolutional Neural Network (CNN) with over 2000 neurons in Sec. V-B. As for equivalence, in Sec. V-C we compute the discrepancy induced by multiple quantization precisions in a neural network for cyber-physical systems, namely a neural controller for the stabilization of an inverted pendulum. Finally, we study how networks, across multiple quantization precisions, may fail satisfying a safety property: we check, in Sec. V-D, for the gender fairness of a predictor for students performance in math exams.

Our benchmarks are publicly available[3]. Our experiments were run on a 2.50GHz Intel Xeon W-2175 CPU, with 64GB memory. We employed TensorFlow Lite for training and quantization, Boolector [20] for bit-precise SMT-solving, and Reluplex [11] for real-numbered networks verification.

A. The effect of quantization upon the robustness against adversarial attacks of a classifier for the MNIST dataset

The MNIST dataset consists of 70,000 handwritten digits represented by 28-by-28 pixel images with a single 8-bit grayscale channel [13]. Each sample belongs to exactly one category $\{0, 1, \ldots 9\}$, which a machine learning model must predict from the raw pixel values. The MNIST set is split into 60,000 training and 10,000 test samples.

We train our neural network as a classifier. In general, a classifier maps a n-dimensional input to one out of m classes ($n = 784$ and $m = 10$, in this case). The chosen class is identified by the output neuron with the largest value: given the output values $z_1, \ldots, z_m$, the choice is given by

$$\mathrm{class}(z_1, \ldots, z_m) = \arg\max_i z_i. \tag{9}$$

The dataset pairs sample inputs s with their respective class c. Training is performed after the training set; then, typically, the quality of the classifier is measured by its *standard accuracy*: the ratio of samples that are correctly classified out of the testing set. Instead, robust accuracy, which we more simply call *robustness*, measures the ratio of robust samples: a sample s with class c is robust when, for some distance $\varepsilon > 0$, it holds that for all perturbations y within that distance the classifier $\mathrm{class} \circ f$ chooses the correct class; in other words, s is robust if the property

$$|s - \vec{y}|_\infty \leq \varepsilon \implies c = \mathrm{class} \circ f(\vec{y}) \tag{10}$$

holds for all $\vec{y}$. Robustness is an assume–guarantee contract, whose guarantee can be encoded as $\bigwedge_{j=1}^m z_j \leq z_c$, where $\vec{z} = f(\vec{y})$ (we additionally assume $z_i \neq z_j$ for all $i \neq j$). A witness for the respective dual satisfiability question (see Eq. 6) constitutes a perturbation for an adversarial attack.

We followed a *post-training quantization* scheme [1]. First, we trained, with floating-point precision (using TensorFlow), a network with 784 inputs, 2 hidden layers of size 64, 32 with ReLU-7 activation function and 10 outputs, for a total of 890 neurons. Afterwards, we quantized the network with 5 to 12 bits and, respectively, in Q3 to Q8 arithmetic, with the exception that we imposed the input layer to be always quantized in 8 bits, the original precision of the samples. We obtained 8 models, whose test (standard) accuracy is shown in Tab. IV, together with that of the floating-point network. On one side of the spectrum, the accuracy values indicate that at least 7 bits are required to obtain an acceptable, i.e., $> 90\%$,

[3]https://github.com/mlech26l/verification_of_quantized_neural_networks

Precision	Std. accuracy
float	94.67%
12 bits	94.64%
11 bits	94.67%
10 bits	94.69%
9 bits	94.38%
8 bits	93.85%
7 bits	91.53%
6 bits	77.39%
5 bits	28.39%

Table IV: Test accuracy of the MNIST classifier.

performance; on the other, 10 bits are enough to match the accuracy of the original network, hence there is no benefit from employing a higher precision. For this reason, we focused our study between the 6 and the 10 bits quantizations.

We checked the robustness of our selected networks on the first 300 test samples of the dataset. To be precise, we checked for the existence of adversarial perturbations within $\varepsilon = 1$ distance on the first 200 test samples and within $\varepsilon = 2$ distance on the next 100. For the quantized networks, we encoded the satisfiability problem (Eq. 6) including the property and the semantics of the network (Eq. 7) and invoked Boolector; for the original network, we invoked Reluplex. For every check, namely for every pair sample–network, we limit the run-time to 24 hours.

As for the bit-precise encoding, we choose all bit-vector variables so to surely avoid overflows. For instance, with a 10 bits quantization each weight w, bias b, and neuron x (except for the input layer) is expressed over 10 bits; moreover, the largest summation involves 64 summands, that is the size of the largest layer. As a result, the largest sum requires at most $20 + \log 64 = 28$ bits; we guarantee this size, for every intermediate variable.

As for the encoding over real-numbers, Reluplex accepts only pure ReLU networks. For this reason, we translate our ReLU-N networks into functionally equivalent ReLU networks, by translating each layer with

$$\text{ReLU-}N(W \cdot \vec{x} + \vec{b}) = \text{ReLU}\Big(-I \cdot \text{ReLU}(-W \cdot \vec{x} - \vec{b} + N)\Big). \quad (11)$$

Out of the 300 samples, at least one method timed out on 56 samples, leaving us with 244 samples whose result was computed over all networks. We show in Tab. V how

Model	Std. Accuracy	Robustness	Median run-time
float/reals	94.67%	91.39%	≈0
10 bits	95.49%	91.39%	8h 58m
9 bits	94.26%	91.39%	5h 34m
8 bits	92.21%	88.93%	3h 41m
7 bits	91.80%	86.48%	1h 29m
6 bits	73.36%	69.67%	18m

Table V: Statistics for the robustness check of 244 MNIST samples.

many of the 244 samples are classified correctly and how many do not accept attacks. The data indicates a discrepancy between standard accuracy and robustness; for real numbered networks, a similar fact was already known in the literature [24]: we empirically confirm that observation for our quantized networks, whose discrepancy fluctuated between 3 and 4% across all precision levels. Besides, while an acceptable, larger than 90%, standard accuracy was achieved at 7 bits, an equally acceptable robustness was achieved at 9 bits.

To obtain all verification results, we spent a total amount of 8009.4 CPU hours (i.e., 334 CPU days). In Fig. 4 we

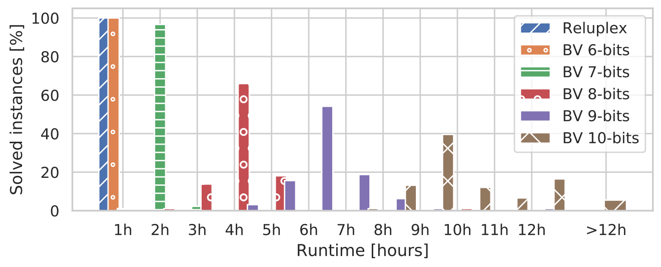


Figure 4: Frequency of run-times for the verification of real and quantized MNIST classifiers.

show the distribution of how long the SMT-solvers took to check the respective robustness properties (values are in percentage and misclassified samples, whose run-time amounts to zero, are not shown). Boolector could tackle—without further optimizations—networks which involved 890 neurons, each; these are already challenging benchmarks for the formal verification task: in the real-numbered world, off-the-shelf solvers could initially tackle up to 20 neurons [6], and modern techniques, while faster, are also evaluated on networks below 1000 neurons, e.g., Reluplex's case study consists of 300 neurons [11], the branch-and-bound's biggest network 894 neurons [18]. On the other hand, Reluplex solved most instanced almost instantly, while Boolector took a median time of 3h 41m and largest time of over 12h. To conclude, it is surprising that the general-purpose solver Boolector, just off-the-shelf, could solve these instances; nevertheless, the performance is not comparable to that of a state-of-the-art specialized solver such as Reluplex. Also, the run-time roughly doubled at every increase in the quantization precision.

B. Convolutional neural networks: computing the robustness of networks beyond thousand neurons

We refined the setup of our previous experiment on MNIST, replacing the two fully-connected layers with a convolutional network architecture consisting of two convolution layers and one fully-connected layer. In particular, we layer 1 made of 8 6x6 filters with 2x2 stride, layer 2 made of 18 6x6 filters with 2x2 stride, and layer 3 made of 64 units, for a total of 2238 neurons; again, the activation functions was a Relu-7.

We train our network according to the *quantization-aware* scheme [2], namely we model the effects of 6-bit quantization during the learning phase. First, we obtained a standard accuracy of 98.56% for the floating-point network, considerably higher of that of Sec. V-A, that was 94.6%. More importantly, we obtained 98.47% standard accuracy for the 6-bit quantized network (against 77.39% of Sec. V-A), practically matching that of its floating-point counterpart. As a result, we expect the network to be also highly robust.

We check the first 400 samples of the test set for adversarial perturbation, both on the real-numbered network and the network quantized with 6 bits. Analogously to Sec. V-A, we translate the first into a pure Relu networks, and give it to Reluplex; we encode the second into a QF_BV formula, and give it to Boolector. We set $\varepsilon = 1$ for the first 200 samples, $\varepsilon = 2$ for the next 200, and check for attacks only the samples that are originally classified correctly; we time-out each check after 24 hours.

ε	Well-classified samples	Attacks	Median run-time	Timeouts
1	198	3	3h 39m	2
2	195	2	5h 31m	34

Table VI: Statistics for the 6-bits quantized CNN.

The results of Tab. VI show that, for the 6-bits networks, we could prove robustness (or compute attacks) for most of the samples. However, the median performance degraded of 10 to 20 times with respect to the 18m of Sec. V-A. Moreover, we could not verify the 7-bits version, as most samples were timing-out; hence, unfortunately, for the current solver these large convolutional networks are out of reach. Notably, Reluplex also failed on the real-numbered version, reporting numerical instability.

C. Functional equivalence of a neural controller with different quantization levels

The verification of cyber-physical system is an important and challenging field of study. However, when neural networks are in the loop, new challenges arise, for which specialized techniques are necessary [25]. Thanks to reinforcement learning, neural networks have been employed in CPS not only for their typical applications such as, e.g., image recognition, but also for automatic control. In this experiment, we analyze a neural controller for the stabilization of an inverted pendulum: we investigate the effect of up- and down-quantization from a reference quantization-aware model.

The goal of an automatic controller for the inverted pendulum is to balance a pole mounted on a cart in an upright position by moving the cart right or left. We train the neural controller in a quantization-aware fashion, optimizing for the 8-bits quantization. We obtain a network, which we call f_8, with 4 input neurons, two hidden layers with respectively 32 and 16 neurons, which altogether feeds to one output neuron; the whole network consists of 53 neurons.

Then, we re-quantize f_8: we down-quantize it to 6- and 7-bits precisions and up-quantize it to 9- and 10-bits precisions, obtaining respectively f_6, f_7, f_9, and f_{10}. To verify equivalence between two networks f_i and f_j, we prove that, for every input $\vec{y}$ constrained within a box (given by the lower and upper bounds $\vec{l}$ and $\vec{u}$), the distance between the outputs is within some $\varepsilon > 0$; this amounts to the validity of

$$\left(\bigwedge_{i=1}^{n} l_i \leq y_i \leq u_i\right) \implies |f_i(\vec{y}) - f_j(\vec{y})| \leq \varepsilon. \tag{12}$$

For every pair of networks, we measure the tightest of such ε, the discrepancy, by binary search over a precision of 0.1.

Our hypothesis is that down-quantized networks differ from f_8, while up-quantized networks are functionally identical to f_8. First of all, we evaluated the performance of all quantizations according to the standard measure (mean return, for $N = 10$). As we show in Tab. VII, all networks quantized

	6 bits	7 bits	8 bits	9 bits	10 bits
Mean return	17.5	67.8	1000.0	1000.0	1000.0

Table VII: Mean return of the quantized neural controllers.

with 8 or more bits solved the task with the top score, while the less precise network failed to do so. At a glance, this confirms our hypothesis. Nevertheless, we need formal analysis to have a definitive answer.

The input space of the inverted pendulum task consists of four variables providing position and velocity of the cart, angle and angular velocity of the pendulum. We construct a set of initial states S_{init} representing the initial position for the problem: the cart is placed in the center, the pendulum is non-deterministically displaced. As for down-quantization, the

	6 bits	7 bits	8 bits	9 bits	10 bits
6 bits	0				
7 bits	0.3	0			
8 bits	1.1	0.9	0		
9 bits	1.1	0.9	0.1	0	
10 bits	1.6	0.9	0.1	0.1	0

Table VIII: Pairwise discrepancies, for the initial set S_{init}.

results of Tab. VIII confirmed our hypothesis. However, up-quantized networks presented small discrepancies both with respect to f_8 and with respect to each other.

We performed the same experiment, but constructing an initial set S_{obs} using bounds derived from 20 simulated runs, plus some small margin; notably, $S_{obs} \supset S_{init}$. The results of

	6 bits	7 bits	8 bits	9 bits	10 bits
6 bits	0				
7 bits	0.4	0			
8 bits	1.3	1.0	0		
9 bits	1.3	1.0	< 1 (oot)	0	
10 bits	> 1 (oot)	> 0.5 (oot)	oot	oot	0

Table IX: Pairwise discrepancies, for the initial set S_{obs}.

Tab. IX did not change our previous observations, when we could measure the discrepancy: as several instances timed-out (after 24h), we could either compute upper or lower bounds for the discrepancy, or nothing at all.

In conclusion, we showed the applicability or our method in equivalence checking, but also its sensitivity to the init set in terms of performance. Besides, our experiment demonstrated that up-quantization does not necessarily preserve functional equivalence, question for which formal analysis was necessary.

D. Fairness in machine learning: quantifying the bias of a neural network

In recent years, machine learning systems have become standard in predicting behavior from large scale historic data such as ranking job candidates, and giving customer recommendations from personalized data. A concern has been raised that decisions of a ML system could discriminate towards certain groups due to a bias in the training data. Consequently, the fairness concern has emerged as a research topic in machine learning [26]. A key issue in quantifying fairness is that neural networks are black-boxes, that is that one cannot explain why neural networks take certain decisions.

We trained a network after a publicly available dataset consisting of 1000 students' personal information and academic test scores [27]. The personal features include gender, ethnic group, parental level of education, reduced vs. standard meal, whether the student took a preparation course for the test, all of which are discrete categorical variables. The scores consist of reading, writing, and math scores, ranging from 0 to 100 in integer values. We train a predictor for students' math scores. Notably, the dataset contains a potential source for gender bias: the mean math score among females is 63.63, among males is 68.73.

The network we trained is composed of 11 input variables, 2 hidden layers with respectively 64 and 32 units, and 1 output variable, for 108 neurons in total. We use an 8-bit quantization-aware training scheme, achieving a 0.45 mean absolute error, i.e., the difference between predicted and actual grade, on the test set.

A fair network is a network for which the grade bias is always within some acceptable bias β; in other words, we verify that

$$\bigwedge_{i \neq \text{gender}} s_i = t_i \wedge s_{\text{gender}} \neq t_{\text{gender}} \implies |f(\vec{s}) - f(\vec{t})| \leq \beta, \quad (13)$$

is valid over the variables $\vec{s}$ and $\vec{t}$, which respectively model two students for which gender differs but all other features are identical—we call them twin students. When we encode the dual formula, we encode two copies of the semantics of same network: to one copy we give one student $\vec{s}$ and take the respective grade g, to the other we give its twin $\vec{t}$ and take grade h; precisely, we check for the satisfiability of

$$\bigwedge_{i \neq \text{gender}} s_i = t_i \wedge s_{\text{gender}} \neq t_{\text{gender}}$$
$$\wedge f(\vec{s}) = g \wedge f(\vec{t}) = h \wedge |g - h| > \beta. \quad (14)$$

Then, we compute a tight upper bound for the bias, that is the maximum possible change in predicted score for any two twin. To compute the tightest bias, we progressively increase β until the formula in Eq. 14 becomes unsatisfiable. In a way, we search for the most adversarial attack, in terms of the set of students' features that induce the largest bias.

We measure mean test error and gender bias of the 6- to the 10-bits quantization of the networks. We show the results in Tab. X. The test error was stable between 4 and 4.5% among all quantizations, showing that the change in precision did not affect the quality of the network in a way that was perceivable by standard measures. However, our formal analysis confirmed a gender bias in the network, producing pairs of twins with a 15 to 21 bias (across quantizations), out of a score span of 100. Surprisingly, the bias monotonically increased as the precision level in quantization lowered, indicating to us that quantization may play a role in determining a bias.

Quantization level	Mean test error	Tightest bias upper bound
6 bits	4.48	21
7 bits	4.16	21
8 bits	4.32	16
9 bits	4.32	15
10 bits	4.64	15

Table X: Results for the formal analysis of the gender bias of a students' grade predictor.

VI. Conclusion

We introduced the first verification method for quantized neural networks which, by SMT solving over bit vectors, accounts for their bit-precise semantics. We showed that bit-precise reasoning is necessary for assessing robustness to adversarial attacks, since verifying the real network can derive false conclusions, both regarding the absence and presence of attacks. We built a tool based on Boolector and evaluated the effect of quantization upon neural networks, computing a networks' robustness against adversarial attacks, but also studying behavioral equivalence between quantized networks and checking the effect upon the validity of safety properties. As for adversarial attacks, we examined several quantizations of a classifier for the MNIST dataset and observed that, statistically, robustness suffers from quantization, proportionally to standard accuracy. Concerning equivalence, we compared different quantizations of a neural controller for an inverted pendulum, confirming the expected large effect of down-quantization, but also discovering a small effect induced by up-quantization. Regarding safety, we measured the gender bias emerging from a predictor for student grades in math exams: not only did we confirm the phenomenon, but also found that it is monotonically enlarged by quantization. Besides, we verified quantized networks up to hundreds of neurons using an off-the-shelf solver. Our run-times were orders of magnitudes larger than those of Reluplex for similar, but real-numbered problems. This poses a limit for current solvers, but also an encouraging baseline for future research.

Acknowledgments

This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23(RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).

References

[1] Post-training quantization. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization

[2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*. IEEE Computer Society, 2018, pp. 2704–2713.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[4] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust physical-world attacks on deep learning models," *arXiv preprint arXiv:1707.08945*, vol. 1, 2017.

[5] L. Schönherr, K. Kohls, S. Zeiler, T. Holz, and D. Kolossa, "Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding," in *accepted for Publication, NDSS*, 2019.

[6] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *CAV*, ser. Lecture Notes in Computer Science, vol. 6174. Springer, 2010, pp. 243–257.

[7] W. Xiang, H. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 29, no. 11, pp. 5777–5783, 2018.

[8] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "AI2: safety and robustness certification of neural networks with abstract interpretation," in *IEEE Symposium on Security and Privacy*. IEEE, 2018, pp. 3–18.

[9] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," in *POPL*. ACM, 2019.

[10] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *CVPR*. IEEE Computer Society, 2016, pp. 2574–2582.

[11] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *CAV (1)*, ser. Lecture Notes in Computer Science, vol. 10426. Springer, 2017, pp. 97–117.

[12] L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Commun.*, vol. 25, no. 2, pp. 117–135, 2012.

[13] The MNIST database. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[14] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV (1)*, ser. Lecture Notes in Computer Science, vol. 10426. Springer, 2017, pp. 3–29.

[15] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *ATVA*, ser. Lecture Notes in Computer Science, vol. 10482. Springer, 2017, pp. 269–286.

[16] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," 2018.

[17] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NFM*, ser. Lecture Notes in Computer Science, vol. 10811. Springer, 2018, pp. 121–138.

[18] R. R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," in *NeurIPS*, 2018, pp. 4795–4804.

[19] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *AAAI*. AAAI Press, 2018, pp. 6615–6624.

[20] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0," *JSAT*, vol. 9, pp. 53–58, 2014.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*. Omnipress, 2010, pp. 807–814.

[22] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, 2010.

[23] ReLU-6. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/nn/relu6

[24] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," in *International Conference on Learning Representations*, 2019.

[25] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," in *NFM*, ser. Lecture Notes in Computer Science, vol. 10227, 2017, pp. 357–372.

[26] S. Barocas, M. Hardt, and A. Narayanan, "Fairness in machine learning," in *Proceeding of NIPS*, 2017.

[27] Students performance in exams. [Online]. Available: https://www.kaggle.com/spscientist/students-performance-in-exams